Discrete Event Dyn Syst (2015) 25:497–529
DOI 10.1007/s10626-014-0195-5

# Model approximation for batch flow shop scheduling with fixed batch sizes

**W. Samuel Weyerman · Anurag Rai · Sean Warnick**

Received: 9 January 2012 / Accepted: 30 April 2014 / Published online: 7 June 2014


**Abstract** Batch flow shops model systems that process a variety of job types using a fixed infrastructure. This model has applications in several areas including chemical manufacturing, building construction, and assembly lines. Since the throughput of such systems depends, often strongly, on the sequence in which they produce various products, scheduling these systems becomes a problem with very practical consequences. Nevertheless, optimally scheduling these systems is NP-complete. This paper demonstrates that batch flow shops can be represented as a particular kind of heap model in the max-plus algebra. These models are shown to belong to a special class of linear systems that are globally stable over finite input sequences, indicating that information about past states is forgotten in finite time. This fact motivates a new solution method to the scheduling problem by optimally solving scheduling problems on finite-memory approximations of the original system. Error in solutions for these "$t$-step" approximations is bounded and monotonically improving with increasing model complexity, eventually becoming zero when the complexity of the approximation reaches the complexity of the original system.

**Keywords** Model approximation · Batch flow shop · Scheduling · Max-plus

W. S. Weyerman
Google, 1600 Amphitheatre Parkway Mountain View, CA, 94043, USA

A. Rai
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

S. Warnick (✉)
Information and Decision Algorithms Laboratories, Department of Computer Science, Brigham Young University, Provo, UT, 84602, USA
e-mail: sean.warnick@gmail.com

## 1 Introduction

Production systems have a rich history and employ a variety of mathematical models to understand their dynamics. The batch flow shop model is a general model class that represents the production of multiple products, generally referred to a job types, using a fixed infrastructure, or set of resources or machines. The complexities of these systems make it difficult to characterize and analyze these models. Nevertheless, the batch flow shop model is useful for several applications, including flowshops, chemical processing plants, and various services. Some specific examples of these application areas are pharmaceutical production, propellant manufacturing, building construction, and assembly lines.

One of the key questions one would like to understand in a production system is how to optimally schedule factory resources when processing a suite of job types. This turns out to be a hard problem. Nevertheless, a variety of models and simplifications have been studied to gain insight about this problem, including exact methods attempting to solve large integer programs (Rich and Prokopakis 1986; Kondili et al. 1993) and the scheduling of a single machine (French 1982; Parker 1995; Pinedo 2005). Many different objectives are specified in Pinedo (2005); these objectives introduce ideas such as minimum makespan, maximum throughput, minimum (weighted) tardiness, minimum lead time, and so on. In van Eekelen et al. (2006) and Boccadoro and Valigi (2003) an optimal 2-product cyclical schedule is determined for a single machine with setup costs. A natural extension to this problem is that of scheduling a job shop with multiple machines. The problem of scheduling several jobs on a set of machines with infinite queues is covered extensively in Pinedo (2005). Other models include a method of treating manufacturing systems as continuous systems in time and controlling them using linear programming is given in van Eekelen et al. (2005), and the so-called "hot ingot" problem, or continuous processing flowshop, which requires a scheduled job to move to the next machine in its route with no delay. The "hot ingot" problem is shown to be equivalent to a single machine with sequence dependent setup times in Reddi and Ramamoorthy (1972). When sequence dependent setup times are added to the flowshop, the sequencing problem is shown to be equivalent to a traveling salesman problem in Gupta (1986). Savkin (2003) considers scheduling a "flexible manufacturing" system, where multiple job types are processed on the same set of machines. Nevertheless, in this work, unbounded buffers are included between each machine, effectively decoupling the interprocessing dynamics.

A more complex environment, and the one we consider, is that with batching machines. Batching machines may process several jobs of a given type simultaneously, but each machine has a fixed batch capacity. Scheduling these systems breaks down into two main problems that need to be solved: lot sizing and scheduling. The lot sizing problem is to determine how many jobs to process at a given time and is discussed in Deng et al. (2002), Dupont and Dhaenens-Flipo (2002) and Glass et al. (2001). The sequencing problem is to determine the order in which to process the jobs to optimize some objective. For our work, we assume that the lot sizing problem has been solved, meaning the capacities of the machines is determined a priori. Since lot sizes are fixed, solving the sequencing problem is all that is needed to solve the scheduling problem, and we will therefore use these terms interchangeably. As with the job shop, there has been considerable effort studying situations where one or two batching machines are present, such as Cheng et al. (2004), Hochbaum and Landy (1997), Lin and Cheng (2001) and Dobson and Nambimadom (2001). In Cheng et al. (2004) a variation of the single machine batch scheduling problem is shown to be NP-Hard. A branch and bound algorithm for lot-sizing and scheduling of a single batch machine is given in Jordan and Drexl (1998). A two-machine factory is considered in Ahmadi et al. (1992)

where one or both machines is a batch machine. These authors consider the sequencing of jobs in the system using full batches and permutation schedules. However, they are limited to two machines and only one machine per workstation. They also simplify the problem by considering a batch machine that has a constant running time for any batch of jobs. Graphical solutions to the general multipurpose batch plant are given in Sanmartí et al. (2002), Rich and Prokopakis (1986) and Kondili et al. (1993), but since these methods are exact, they are also computationally intractable for a complex manufacturing system. Two heuristic methods of solution to the minimum makespan problem are given in Blömer and Günther (1998), the better of the two reduces decision variables in the MILP by a linear factor. Although this does allow for the solution of more complex problems, it is still difficult to compute the makespan for a manufacturing system containing a workstation with a much larger capacity or longer processing time than another workstation. Other approaches to scheduling include solving a stochastic integer programming problem, Sand and Engell (2004), and a hybrid constraint logic programming (CLP) and mixed integer linear programming (MILP) approach, Roe et al. (2005). Bemporad (2003) and Corona et al. (2005) consider optimization in the face of quantization and partition the state space offline to facilitate fast decision making.

Much of the analysis and modeling of job shops and manufacturing systems is done using discrete event systems. Multipurpose batch manufacturing systems are modeled as Petri nets in Riera et al. (2005), Yajima et al. (2004), López-Mellado et al. (2005). An optimization procedure for Petri nets is also given in Riera et al. (2005), however, it is exact and thus not tractable for complex systems, so they offer a tree pruning heuristic to reduce the search time. A method for representing Petri nets as heaps of pieces in the max-plus algebra is given in Gaubert and Mairesse (1999), and much of our work is built on this foundation. Certain discrete event processes are shown to be linear in the max-plus algebra in Cohen et al. (1985) and Baccelli et al. (1992). They note that this is useful for performance evaluation in manufacturing. Other approaches to scheduling using max-plus models include Yurdakul and Odrey (2004), Bouquard et al. (2006a), Bouquard et al. (2006b), and Van den Boom and De Schutter (2006).

In this work we show that a batch flow shop can be modeled as an input quantized system in the max-plus algebra. This development is done by modeling the system as a heap of "non-rigid" pieces, in stark contrast to previous work using heaps of "rigid blocks". Doing this, we show that the batch flow shop models are a subset of a larger class of systems in the max-plus algebra, a class that can be shown to be globally stable over finite-length inputs. This stability property causes these systems to forget initial condition information in a finite number of steps and motivates our approximation method. Our approach is to optimally solve the sequencing problem for a simplified model of the batch flow shop (Beck et al. 1996; Dullerud and Paganini 2000). The organization of this paper is as follows. We first present the max-plus algebra. This allows us to introduce the batch flow shop model and represent it as a max-plus dynamical system. We show that batch flow shops are a subset of a larger class of systems in the max-plus algebra that exhibit a particular structure. We then show that any system in this class exhibits a form of stability. We finally present our method of model approximation over this class of systems and show that this method has bounded error.

## 2 Max-plus algebra preliminaries

This work draws from the standard *heaps of pieces* or *heap model* that is described using the max-plus algebra, see for example Gaubert and Mairesse (1999), Heidergott et al. (2006).

These models describe situations like the popular Tetris game, where pieces of various shapes and orientations pile up vertically. This stacking mechanism is conveniently described as a linear system defined over the max-plus (or sometimes called *tropical*) semiring, described briefly below, along with definitions of system stability and various performance measures.

### 2.1 Background and notation

**Definition 1** The (max, +) semiring, $\mathbb{R}_{\max}$, is the set $\mathbb{R} \cup \{-\infty, +\infty\}$, equipped with the operations

1. $\oplus$, where $a \oplus b := \max(a, b)$,
2. $\otimes$, where $a \otimes b := a + b$,

where $\mathbb{R}$ is the set of real numbers. Note that often the symbol $\overline{\mathbb{R}}_{\max}$ is used to indicate the complete (max, +) semiring, defined over $\mathbb{R} \cup \{-\infty, +\infty\}$, but in this work, where there should be no confusion about the meaning, we remove the overline to simplify notation. Also, note that this semiring is idempotent, since $x \oplus x = x$ for all $x \in \mathbb{R}_{\max}$; the zero element is negative infinity, denoted $\epsilon$; and the unit element is zero, denoted $e$. Moreover, in this work, we will further consider the multiplicative inverse operation, $\oslash$, where $a \oslash b := a - b$ for all $\{a, b\} \in \mathbb{R}_{\max}$, making $\mathbb{R}_{\max}$ a semifield. Throughout this paper we will use the convention $\epsilon \oslash \epsilon = \epsilon$, suggesting that for any $y \in \mathbb{R}_{\max}$, the indeterminate form $y \oplus (\epsilon \oslash \epsilon) = y$.

Matrix arithmetic is also defined over the (max,+) semiring. For matrices, $A, B \in \mathbb{R}_{\max}^{n \times l}$, $C \in \mathbb{R}_{\max}^{l \times m}$ we define the operations:

$$[A \oplus B]_{ij} := a_{ij} \oplus b_{ij}$$
$$[B \otimes C]_{ik} := \bigoplus_{j=1}^{l} \left(b_{ij} \otimes c_{jk}\right).$$

The zero vector, $\boldsymbol{\epsilon}$, and the unit vector, $\mathbf{e}$, are given by

$$\boldsymbol{\epsilon} := \begin{bmatrix} \epsilon \\ \vdots \\ \epsilon \end{bmatrix}, \qquad \mathbf{e} := \begin{bmatrix} e \\ \vdots \\ e \end{bmatrix},$$

where $e$ and $\epsilon$ are as defined above. The identity matrix is

$$I_{max} := \begin{bmatrix} e & \epsilon & \dots & \epsilon \\ \epsilon & e & & \epsilon \\ \vdots & & \ddots & \vdots \\ \epsilon & \epsilon & \dots & e \end{bmatrix}.$$

Spectral theory is well developed over the max-plus semiring. The following definition will be useful in the subsequent development in later sections.

**Definition 2** (Heidergott et al. 2006) Let $A \in \mathbb{R}_{\max}^{n \times n}$ be a square matrix. If $\mu \in \mathbb{R}_{\max}$ is a scalar and $\mathbf{v} \in \mathbb{R}_{\max}^{n}$ is a vector that contains at least one finite element such that

$$A \otimes \mathbf{v} = \mu \otimes \mathbf{v},$$

then $\mu$ is called an *eigenvalue* of $A$ and $\mathbf{v}$ an *eigenvector* of $A$ associated with eigenvalue $\mu$.

We can now define a linear state-space system in the max-plus algebra. For $\mathbf{x}(k) \in \mathbb{R}^n_{\max}$, $k \in \mathbb{N}$, where $\mathbb{N}$ denotes the non-negative integers, and $A \in \mathbb{R}^{n\times n}_{\max}$, the sequence generated by the recursion,

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k), \tag{1}$$

is well-defined and characterized by

$$\mathbf{x}(k) = A^{\otimes k}\mathbf{x}_0,$$

where $\mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^n_{\max}$ is the initial condition, and $A^{\otimes k}$ denotes the matrix $A$ raised to the $k^{th}$ power, in the max-plus sense of matrix multiplication.

Similarly, if the system is time-varying, such as

$$\mathbf{x}(k+1) = A(k) \otimes \mathbf{x}(k), \tag{2}$$

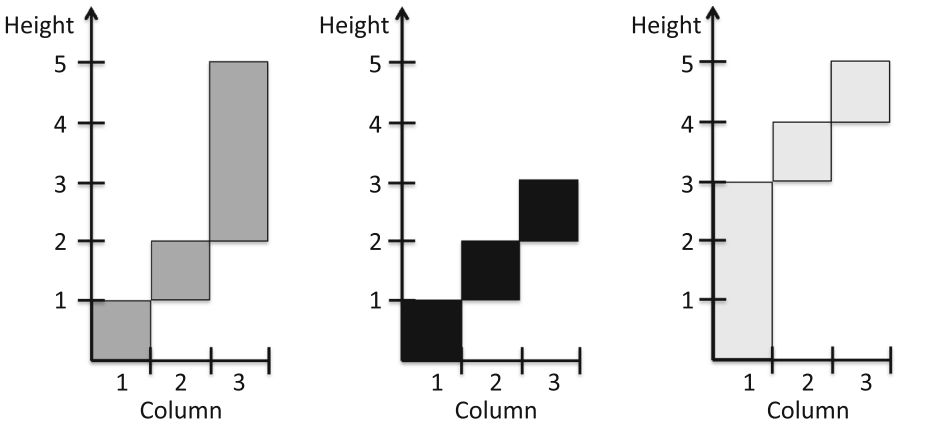
the solution is well-defined and characterized by

$$\mathbf{x}(k) = \left(\bigotimes_{i=0}^{k-1} A(i)\right)\mathbf{x}_0,$$

where $\mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^n_{\max}$ is the initial condition.

## 2.2 Rigid-block heap models

These max-plus linear systems, such as that in Eq. 2), are the basis for a standard *heap model* (Gaubert and Mairesse 1999) that describes the surface of a pile of predefined, rigid blocks, or pieces. Like the game Tetris, imagine a stack of blocks that have piled up vertically, as in Fig. 2. Unlike the Tetris game, however, these blocks do not rotate or move horizontally; each block is characterized by a well-defined shape over specific columns. We associate, then, each block with a particular matrix, $A(i) \in \mathbb{R}^{n\times n}_{\max}$, $i = 1, \ldots, m$, where $m$ is the total number of distinct block types, and the vector $\mathbf{x}(k) \in \mathbb{R}^n_{\max}$ describes the height of the stack in each of the $n$ columns after the $k^{th}$ block is added to the pile. The max-plus linear system (2) computes the changing surface vector, $\mathbf{x}$ as each new block is added (Fig 1).




(a) Shape of block a (b) Shape of block b (c) Shape of block c

**Fig. 1** Block shapes for 3-block heap system example

For example, consider the 3-block system characterized in Table 1. In this system, there are three distinct shapes of blocks, labeled $a$, $b$ and $c$, constructed over three columns. Two vectors in $\mathbb{R}^3_{\max}$ characterize each block by defining its upper contour, $\mathbf{u}(p)$, and lower contour, $\mathbf{l}(p)$, and these contour vectors are then used to construct a matrix, $A(p)$, associated with each block type, $p \in \{a, b, c\}$, as follows:

$$[A(p)]_{ij} = \begin{cases} \mathbf{u}_i(p) - \mathbf{l}_j(p) & \text{if } i, j \in R(p) \\ e & \text{if } i = j, i \notin R(p) \\ \epsilon & \text{otherwise} \end{cases} \tag{3}$$

where $R(p)$ is the set of columns occupied by block type $p$.

Starting from an "empty floor," $\mathbf{x}(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, the changing surface of the stack of blocks, or heap, driven by the input sequence a,b,c is then captured by the dynamics described in Eq. 2 as follows:

$$\begin{bmatrix} \mathbf{x}_1(1) \\ \mathbf{x}_2(1) \\ \mathbf{x}_3(1) \end{bmatrix} = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 5 & 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{x}_1(2) \\ \mathbf{x}_2(2) \\ \mathbf{x}_3(2) \end{bmatrix} = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{x}_1(3) \\ \mathbf{x}_2(3) \\ \mathbf{x}_3(3) \end{bmatrix} = \begin{bmatrix} 3 & e & -1 \\ 4 & 1 & e \\ 5 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}.$$

This sequence of blocks a,b,c, results in the surface vector $\mathbf{x}(3) = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}^T$, as illustrated in Fig. 2.

### 2.3 Stability and performance measures

Throughout this paper we will be interested in the behavior of max-plus linear systems, such as that in Eq. 2, for large $k$. The following definitions will help clarify the resulting analysis (Table 2).

**Table 1** Mathematical modeling of a 3-block system

| Block | Upper contour | Lower contour | Matrix representation |
|---|---|---|---|
| a | $\mathbf{u}(a) = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$ | $\mathbf{l}(a) = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$ | $A(a) = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 5 & 4 & 3 \end{bmatrix}$ |
| b | $\mathbf{u}(b) = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ | $\mathbf{l}(b) = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$ | $A(b) = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix}$ |
| c | $\mathbf{u}(c) = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$ | $\mathbf{l}(c) = \begin{bmatrix} e \\ 3 \\ 4 \end{bmatrix}$ | $A(c) = \begin{bmatrix} 3 & e & -1 \\ 4 & 1 & e \\ 5 & 2 & 1 \end{bmatrix}$ |

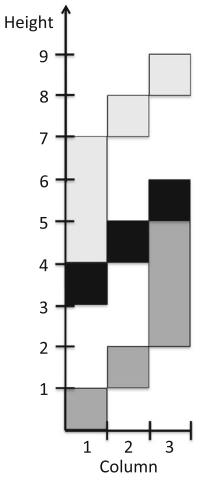



**Fig. 2** Heap resulting from stacking blocks **a**, **b**, and **c**

**Definition 3** The system given by Eq. 1 and characterized by the matrix $A \in \mathbb{R}_{\max}^{n\times n}$ is said to be *globally stable* if, for any initial condition $x(0)$:

$$\exists N \in \mathbb{N} \text{ and unique } v_1, ..., v_n \in \mathbb{R} \text{ such that } k \geq N \implies x_i(k) \oslash x_1(k) = v_i,$$

where $\mathbb{N}$ is understood to be the set of natural numbers.

Global stability implies that, after a finite number of steps, the relative distance among the elements of $\mathbf{x}(k)$ remains fixed as $k \to \infty$. This suggests a notion of convergence in a projective space, formed as the quotient space of $\mathbb{R}_{\max}^n$ by a particular equivalence relation (see Section 1.4 in Heidergott et al. (2006) for details). In particular, while the values of the elements of the state vector of a globally stable system may grow without bound, the relative differences among elements of $\mathbf{x}$ remain constant after some initial transient period. Moreover, this limiting behavior of $\mathbf{x}$ is unique, independent of the initial condition.

In addition to the limiting behavior characterized by global stability, we will also be interested in various measures characterizing the quality of our approximations. In particular, we will abuse notation and consider the following functions that are not norms, but are reminiscent of these measures and will be useful in the sequel.

**Table 2** Comparing rigid-block heap matrices with flow shop heap matrices

| Block | Upper contour | Lower contour | Rigid-block matrix | Flow shop matrix |
|---|---|---|---|---|
| a | $\mathbf{u}(a) = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$ | $\mathbf{l}(a) = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$ | $A(a) = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 5 & 4 & 3 \end{bmatrix}$ | $A(a) = \begin{bmatrix} 1 & e & \epsilon \\ 2 & 1 & e \\ 5 & 4 & 3 \end{bmatrix}$ |
| b | $\mathbf{u}(b) = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ | $\mathbf{l}(b) = \begin{bmatrix} e \\ 1 \\ 2 \end{bmatrix}$ | $A(b) = \begin{bmatrix} 1 & e & -1 \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix}$ | $A(b) = \begin{bmatrix} 1 & e & \epsilon \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix}$ |
| c | $\mathbf{u}(c) = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$ | $\mathbf{l}(c) = \begin{bmatrix} e \\ 3 \\ 4 \end{bmatrix}$ | $A(c) = \begin{bmatrix} 3 & e & -1 \\ 4 & 1 & e \\ 5 & 2 & 1 \end{bmatrix}$ | $A(c) = \begin{bmatrix} 3 & e & \epsilon \\ 4 & 1 & e \\ 5 & 2 & 1 \end{bmatrix}$ |

**Definition 4** The 1-measure of a max-plus vector, $\mathbf{b} \in \mathbb{R}^n_{\max}$ is

$$||\mathbf{b}||_{1_{max}} = \bigoplus_{i=1}^{n} b_i = \mathbf{e}^T \otimes \mathbf{b}.$$

Note that this function returns the maximum element of a vector. Moreover, this function induces a similar function on a matrix.

**Definition 5** The max-plus 1-induced measure of a matrix $A \in \mathbb{R}^{n\times m}_{\max}$ is

$$||A||_{1_{max}} = \max_{\mathbf{x}\in\mathbb{R}^{m\times 1}_{\max}, \mathbf{x}\neq\epsilon} (||A \otimes \mathbf{x}||_{1_{max}} \oslash ||\mathbf{x}||_{1_{max}}).$$

**Lemma 1** *Given a matrix $A \in \mathbb{R}^{n\times m}_{\max}$, the max-plus 1-induced measure of A is equivalent to its largest element:*

$$||A||_{1_{max}} = \max_{ij} a_{ij}.$$

*Proof* Let $A \in \mathbb{R}^{n\times m}_{\max}$ be given. Without loss of generality, we will say that $||\mathbf{x}||_{1_{max}} = e$. Note that $||A \otimes \mathbf{x}||_{1_{max}} = \mathbf{e}^T \otimes A \otimes \mathbf{x}$. The vector $v^T =: \mathbf{e}^T \otimes A$ is the vector containing the max element of each column of $A$. Therefore, we want to maximize $v^T \otimes \mathbf{x}$. Because $||\mathbf{x}||_{1_{max}} = e$, the largest element in $\mathbf{x}$ is $e$. To maximize $v^T \otimes \mathbf{x}$, we want to make each element of $\mathbf{x}$ as large as possible; this means we set $\mathbf{x} = \mathbf{e}$ which gives $v^T \otimes \mathbf{x} = \max_{ij} a_{ij}$. □

By this theorem, we see that the max-plus 1-induced measure of a matrix is very simple to compute. We are also interested in a similar quantity, $\min_{\mathbf{x}} \|A \otimes \mathbf{x}\|_{1_{max}} \oslash \|\mathbf{x}\|_{1_{max}}$. This quantity is also simple to compute.

**Lemma 2** *Given a matrix $A \in \mathbb{R}^{n\times m}_{\max}$,*

$$\min_{\mathbf{x}\in\mathbb{R}^{m\times 1}_{\max}, \mathbf{x}\neq\epsilon} \left(||A \otimes \mathbf{x}||_{1_{max}} \oslash ||\mathbf{x}||_{1_{max}}\right) = \min_i \left[\mathbf{e}^T \otimes A\right]_i.$$

*Proof* Let $A \in \mathbb{R}^{n\times m}_{\max}$ be given. Without loss of generality, let $||\mathbf{x}||_{1_{max}} = e$ and consider $v^T \otimes \mathbf{x}$ with $v^T =: \mathbf{e}^T \otimes A$. Now we want to minimize $v^T \otimes \mathbf{x}$, so we want each element of $x$ as small as possible. However, having $||\mathbf{x}||_{1_{max}} = e$ requires at least one element of $\mathbf{x}$ equal to $e$. Thus, we need only consider each $\mathbf{x}$ where $\mathbf{x}_i = e$, and $\mathbf{x}_j = \epsilon$ for $j \neq i$. So

$$\begin{aligned}\min_{\mathbf{x}} \left[v^T \otimes \mathbf{x}\right] &= \min_i (v_i) \\ &= \min_i \left[\mathbf{e}^T \otimes A\right]_i.\end{aligned}$$

□

## 3 Model for a batch flow shop

Consider a batch flow shop model given by $n$ workstations that operates on $m$ distinct job types. As a flow shop, each job type follows the same route through the system and workstations are ordered accordingly. We restrict our attention to the situation where there

is no intermediate storage between the machines (however the machines themselves can store the load whenever necessary) and each workstation is composed of a single machine.

We focus on the sequencing problem by assuming that the lot sizing problem has previously been solved. This defines a fixed capacity for each job type, $j$, on each workstation $i$, $c_{ij}$. Likewise, each job type has a fixed processing time on each workstation, $\tau_{ij}$, yielding the recipe $(\mathbf{c}, \boldsymbol{\tau})(j)$ for every job type $j \in \mathscr{J} = \{1, \ldots, m\}$.

Fixed routes, no intermediate storage, and fixed capacities restrict admissible schedules for the system to be permutation schedules. That is, each workstation must follow the same sequence of job types. Furthermore, by imposing a non-idling policy, every sequence uniquely specifies an admissible schedule for the system.

The fact that the capacity of the first workstation is fixed for each job type implies that commanding a particular job type requires at least the number of jobs of this type to fill the capacity of the first workstation. Moreover, since other workstations may have different capacities for the same job type, more than one batch may be required of the first workstation. For example, if capacity of the second workstation is twice that of the first, then commanding this job type will require at least two batches from the first workstation. The least common multiple of the workstation capacities for a given job type $j$, called the load, $L(j)$, is the minimum number of jobs of this type that must be commanded to satisfy the workstation capacities and admissibility requirements of the system. From this we see that workstation $i$ must process $B(i) = L(j)/c_{ij}$ batches of job type $j$ to complete a load.

These requirements imply that the scheduling problem consists of choosing a sequence of loads of various job types to meet a specified quota $\mathbf{q} \in \mathbb{N}^m$, where $\mathbb{N}$ is the set of natural numbers. Let $Q = \sum_{i=1}^{m} q_i$. Only quotas that are integer multiples of loads for each job type make sense. This sequence of loads, $u_k \in \mathscr{J}, k = 0, \ldots, Q$, is the input to the system, which we will represent as evolving with load-events, $k$, rather than in real time.

The state of the system, $\mathbf{x}(k) \in \mathbb{R}^n$, represents the time at which each workstation is available for processing the $k^{\text{th}}$ load. The system output, $y(k)$, represents the transition time from state $\mathbf{x}(k)$ to $\mathbf{x}(k+1)$, which is the time it takes to finish the $(k+1)^{th}$ load after finishing the $k^{th}$ load. We note that the dynamics of this system are not linear in the convential algebra. As a result, we will characterize this system as a discrete-time dynamical system with linear dynamics (albeit a nonlinear output function) over the max-plus semiring as a modified heap model.

### 3.1 Max-plus representation

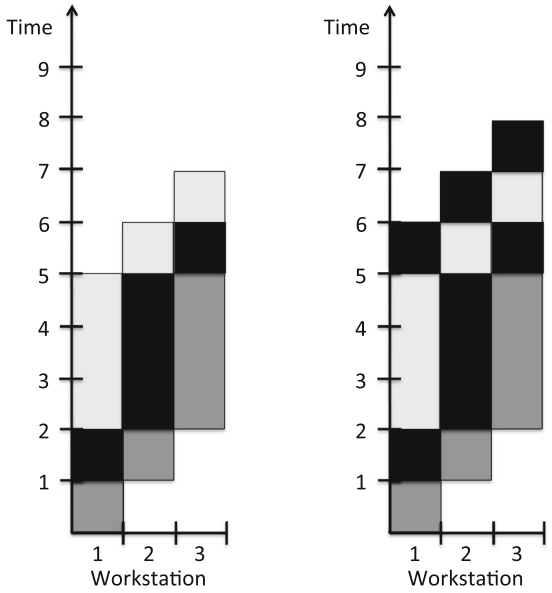
The heap model discussed previously makes a nearly ideal model for the flow shop described above. In the heap model, blocks are defined by their shape, characterized by height over a fixed set of columns. In the flow shop, each load is also defined by a shape, characterized by the amount of time it requires from each workstation, over a fixed set of workstations. We see, then, that the state of the system, $\mathbf{x}(k) \in \mathbb{R}^n_{\max}$, representing the time when each of the $n$ workstations is available for processing the $k^{th}$ load, evolves as Eq. 2, where each job type $j \in \mathscr{J} = \{1, \ldots, m\}$ is associated with a matrix $A(j) \in \mathbb{R}^{n \times n}_{\max}$. An empty flow shop begins from $\mathbf{x}(0) = \mathbf{e}$ and processes $Q$ loads, defining a sequence $\mathbf{x}(k)$, $k = 0, ..., Q-1$, to meet its quota.

There are some important differences, however, between the rigid-block heap model discussed previously, and a heap model capable of describing the dynamics of the flow shop described above. First, in the flow shop considered here, a non-idling policy is enforced. This ensures that every sequence of loads generates a unique admissible schedule, thereby reducing the scheduling problem to a sequencing problem. It also implies, however, that if a

machine is available and the next load is ready, then it should start processing the next load, which (along with the no-intermediate-storage policy) can warp the "shape" associated with the next load.

For example, consider Fig. 2. If we associate loads with blocks, columns with workstations, and the height of a block with the time that load requires from each workstation, then we see immediately that a non-idling policy would warp the shape of block, or load, $b$. This is because workstation 1 becomes available after processing its first load (of job-type $a$) at time 1, so it should then immediately begin processing load $b$. Workstation 1 will finish processing load $b$ at time 2, the exact same time when workstation 2 finishes processing load $a$. As a result, it will then unload $b$ into workstation 2, which will immediately begin processing load $b$. Workstation 2 will finish processing load $b$ at time 3. Nevertheless, the no-intermediate-storage policy indicates that Workstation 2 can not unload $b$ until time 5, when Workstation 3 completes its processing of load $a$, further warping the shape of load $b$.

Figure 3 illustrates how the flow shop dynamics considered here would process the three jobs from Table 1. Notice that, unlike Fig. 2, here the blocks representing loads are not rigid. For example, in Fig. 3b we note that, even in the same sequence, the shape of load $b$ can be different, depending on the state of the system when it is processed. This important difference between the flow shop model and the rigid-block heap model enables the flow




(a) Quota $\{a, b, c\}$ completed two time units sooner than the rigid-block heap model shown in Figure 2.

(b) Processing an additional load of $b$ highlights how the shape associated with load $b$ (black piece) changes due to the non-idling and no-intermediate-storage policies of the flow shop, demanding a new heap model.

**Fig. 3** Flow shop heap resulting from processing loads **a**, **b**, and **c** (*left*) and another load of **b** (*right*) from the heap example in Table 1

shop to process a given quota faster than its rigid-block counterpart. For example, in Fig. 3a, the same sequence, $a$, $b$, and $c$, is completed about 22 % sooner than the rigid-block system.

The other important difference between the rigid-block heap model discussed previously, and the flow shop model developed here, is that flow shops are characterized by the recipes of their job types, each represented by a capacity vector and a processing-time vector, $(\mathbf{c}, \boldsymbol{\tau})(j)$ for every job type $j \in \mathscr{J} = \{1, \ldots, m\}$, while blocks are represented by upper and lower contour vectors (see Table 1). The mapping from recipes to contours is not trivial, but it is nevertheless well-defined and systematic, yielding a process for establishing a fixed matrix $A(j) \in \mathbb{R}_{\max}^{n \times n}$ associated with each job type $j$. The remainder of the section develops the flow shop heap model by considering both of these differences in detail.

### 3.2 Non-rigid heaps

Deriving the upper and lower contours characterizing the nominal "shape" of a load for a given job type is not trivial, and this is the topic of the next section. However, if we had these contours, $\mathbf{u}(j)$ and $\mathbf{l}(j)$, we could consider how to develop an appropriate job matrix $A(j)$. This job matrix needs to capture the nominal "shape" of a load for a given job type, but not in the rigid way described previously. Instead, it needs to operate on the system state in a manner consistent with the nominal "shape" of a given load, but also with the non-idling and no-intermediate-storage character of the flow shop.

Surprisingly, these effects, due to non-idling and no-intermediate-storage, can be captured with a slight change to the rigid-block matrices. The key idea is to note that whenever an entry of a rigid-block matrix is negative, this indicates that one workstation completes its processing of a load before another workstation even starts work on the same load. In a flow shop with non-idling and no intermediate storage, the partial decoupling of such machines is captured by replacing the negative entry with $\epsilon$.

$$[A(p)]_{ij} = \begin{cases} u_i(p) - l_j(p) & \text{if } u_i(p) - l_j(p) \geq 0 \text{ and } i, j \in R(p) \\ e & \text{if } i = j, i \notin R(p) \\ \epsilon & \text{otherwise.} \end{cases} \tag{4}$$

where $R(p)$ is the set of workstations used by job-type $p$. Compare this construction procedure with that for the rigid-block heap model in Eq. 3 to verify the replacement of negative entries by $\epsilon$.

By way of comparison, consider the blocks from the rigid-block example discussed previously. It is easy to contrast the block matrices derived for each set of contours and verify that the linear max-plus system (2) generates the surface vectors suggested by Fig. 3.

Starting from an empty flow shop," $\mathbf{x}(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, the changing availability of workstations, driven by the input sequence $p(k) = abc$ for $k = 0, 1, 2$, is then captured by the dynamics described in Eq. 2 as follows:

$$\begin{bmatrix} \mathbf{x}_1(1) \\ \mathbf{x}_2(1) \\ \mathbf{x}_3(1) \end{bmatrix} = \begin{bmatrix} 1 & e & \epsilon \\ 2 & 1 & e \\ 5 & 4 & 3 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{x}_1(2) \\ \mathbf{x}_2(2) \\ \mathbf{x}_3(2) \end{bmatrix} = \begin{bmatrix} 1 & e & \epsilon \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 6 \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{x}_1(3) \\ \mathbf{x}_2(3) \\ \mathbf{x}_3(3) \end{bmatrix} = \begin{bmatrix} 3 & e & \epsilon \\ 4 & 1 & e \\ 5 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix}.$$

This sequence of blocks, $p(k) = abc$ for $k = 0, 1, 2$, results in the state vector $\mathbf{x}(3) = \begin{bmatrix} 5 & 6 & 7 \end{bmatrix}^T$, as illustrated in Fig. 3a. Adding an additional load of $b$ then yields

$$\begin{bmatrix} \mathbf{x}_1(2) \\ \mathbf{x}_2(2) \\ \mathbf{x}_3(2) \end{bmatrix} = \begin{bmatrix} 1 & e & \epsilon \\ 2 & 1 & e \\ 3 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 6 \\ 7 \\ 8 \end{bmatrix},$$

as illustrated in Fig. 3b.

Thus we observe that a variation on the rigid-block heap model offers a new heap model that captures the dynamics of the non-idling, no-intermediate-storage, permutation scheduled flow shop, provided that the recipe for each product type could be successfully translated into an upper contour and lower contour vector characterizing the demand of a load on the various workstations. The next section explores this translation process.

### 3.3 Recipes to contours and heap matrices

The difficulty in translating a recipe for a given job to upper and lower contour vectors arises primarily because of the batched nature of this type of flow shop system. Since the capacity of a particular workstation can be different than a subsequent workstation, multiple batches of the product may need to be processed in order to produce a single load. This, coupled with the no-intermediate-storage policy, allows recipes to couple the demand on workstations in complicated ways, forcing workstations to behave as fixed-capacity queues as well as processors.

For example, consider a recipe characterized by $\mathbf{c} = [6, 2, 3]^T$, $\tau = [6, 1, 1]^T$. A load of this job type begins (see Fig. 4) with a batch of 6 units processed on workstation 1 for 6 time units. Once completed, however, not all of the 6 units can proceed to Workstation 2, since the capacity of Workstation 2 is only 2 units. As a result, Workstation 1 continues to hold 4 units while Workstation 2 processes 2 units. At time 7, Workstation 2 then loads 2 units into Workstation 3. However, Workstation 3 can not begin to process these units until it has reached full capacity of 3 units, so it simply holds these units while Workstation 2 process

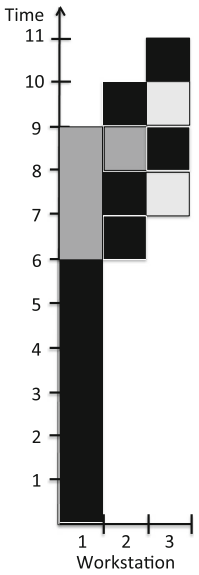



**Fig. 4** The characteristic shape of a single load for a job with recipe $\mathbf{c} = [6, 2, 3]^T$, $\boldsymbol{\tau} = [6, 1, 1]^T$ is the combination of processing (*black*), holding processed material until a downstream workstation is available (*medium grey*), or holding unprocessed material until full capacity is reached (*light grey*). This complicated interaction among workstations makes computing the upper and lower contours for the load nontrivial

a second batch, leaving Workstation 1 holding 2 units. When Workstation 2 completes its second batch, it unloads a single unit to fill Workstation 3 to a (full) capacity of 3 units. Workstation 3 then processes its first batch while Workstation 2 waits by holding 1 unit. At time 9, when Workstation 3 completes its first batch, Workstation 2 unloads the unit it was holding and grabs the last 2 units that Workstation 1 was holding, relieving Workstation 1 of any further obligation. Workstation 2 then processes its last batch, loads it into Workstation 3 which, combined with the single unit that Workstation 3 was holding, makes a full batch for Workstation 3 to process, completing the load.

Because of this difficulty translating recipes into upper and lower contours, the algorithm in Fig. 5 builds the $A$ matrix for the load directly. It accomplishes this by considering the block associated with a load as, itself, a heap of two kinds of pieces, a processing piece and a precedence piece.

**Definition 6** The matrix for the processing piece for job type $j$ with recipe $(\mathbf{c}, \tau)(j)$ at workstation $i$, $P_i(j)$, is constructed from the lower contour

$$\begin{bmatrix} \epsilon & \cdots & e & \cdots & \epsilon \end{bmatrix},$$

where $e$ is the $i^{\text{th}}$ element, and the upper contour

$$\begin{bmatrix} \epsilon & \cdots & \tau_i(j) & \cdots & \epsilon \end{bmatrix},$$

where $\tau_i(j)$ is the $i^{\text{th}}$ element.

**Definition 7** The matrix for the precedence piece for job type $j$ with recipe $(\mathbf{c}, \boldsymbol{\tau})(j)$ at workstation $i$, $R_i(j)$, is constructed from the (equal) upper and lower contours, given by

$$\begin{bmatrix} \epsilon & \cdots & e & e & \cdots & \epsilon \end{bmatrix},$$

where $e$ is the $i^{\text{th}}$ and $i+1^{\text{th}}$ elements.

Therefore the time that workstation $i$ spends processing a batch of job type $j$ is represented as a piece occupying resource $i$ with height $\tau_i(j)$. The matrix, $P_i(j)$, is equal to $I_{max}$ except that $[P_i(j)]_{ii} = \tau_i(j)$. Also, the precedence of workstation $i$ over $i+1$ is represented as a piece occupying resources $i$ and $i+1$ with height 0. This matrix, $R_i$, is equal to $I_{max}$ except that $[R_i]_{i,i+1} = [R_i]_{i+1,i} = e$.

Using these rigid pieces and the algorithm given in Fig. 5 we can construct the non-rigid block for a single load of a job of type $j$. This algorithm is a simulation of one load in the factory. At each iteration, the algorithm checks a machine to see if it is ready to process the load, i.e. the machine does not contain a processed load waiting to be unloaded and the load meets its full capacity, if so, it adds a processing piece $(P)$ by multiplying it to the heap matrix. If the machine has a processed load and it can unload it to the next machine, the algorithm adds a precedence piece $(R)$ to the heap. Finally, this algorithm arrives at a heap matrix for a single load of product type $j$, which we refer to as $A(j)$.

### 3.4 The flow shop model

Using the Recipe-to-Heap algorithm in Fig. 5, we can now identify a particular heap matrix, $A(j) \in \mathbb{R}_{\max}^{n \times n}$ with a load of each job type $j \in \mathcal{J} = \{1, \ldots, m\}$. Noting that we desire

```
procedure RecipeToHeap(c ∈ ℝ^n, τ ∈ ℝ^n)
    M ← I_n
    P ← matrices for processing pieces
    R ← matrices for precedence pieces
    amtAcum ← ldProc ← zeros(n,1)
    ldRequired ← lcm(c) ./ c
    hasProcessedLoad ← zeros(n,1)
    while ldProc ≠ ldRequired do
        for i = 1 : n do
            if hasProcessedLoad[i] = 0 then
                if ldProc[i] < ldRequired[i] then
                    if i = 1 then
                        M ← P[i] ⊗ M
                        amtAcum[i] ← c[i]
                        hasProcessedLoad[i] ← 1
                        ldProc[i] ← ldProc[i] + 1
                    else
                        if amtAcum[i] = c[i] then
                            M ← P[i] ⊗ M
                            hasProcessedLoad[i] ← 1
                            ldProc[i] ← ldProc[i] + 1
                        end if
                    end if
                end if
            end if
            if hasProcessedLoad[i] = 1 then
                if i = n then
                    amtAcum[i] ← 0
                    hasProcessedLoad[i] ← 0
                else
                    if amtAcum[i+1] < c[i+1] and unloading[i+1] ≠ 1 then
                        M ← R[i] ⊗ M
                        amtUnloaded ← min(c[i+1] − amtAcum[i+1], amtAcum[i])
                        amtAcum[i+1] ← amtAcum[i+1] + amtUnloaded
                        amtAcum[i] ← amtAcum[i] − amtUnloaded
                        if amtAcum[i] = 0 then
                            hasProcessedLoad[i] ← 0
                        end if
                    end if
                end if
            end if
        end for
    end while
    return M
end procedure
```

**Fig. 5** Recipe to heap algorithm

the output of our system to measure the processing time for each load, the flow shop model then becomes:

$$\begin{aligned} \mathbf{x}(k+1) &= A(p(k)) \otimes \mathbf{x}(k) \\ y(k) &= \|A(p(k)) \otimes \mathbf{x}(k)\|_{1_{max}} \oslash \|\mathbf{x}(k)\|_{1_{max}}, \end{aligned} \tag{5}$$

where $p(k) \in \mathscr{J} = \{1, \ldots, m\}$ is the input sequence of job-types. Note that this system has linear dynamics, albeit a nonlinear output function.

The real advantage of this model, however, comes from the fact that all admissible heap matrices have a particular structure that guarantee certain important properties. Defining $\xi$ to be the column difference of $A$ as:

$$\xi_{ij}(A) = a_{ij} - a_{i,j+1},$$

we then have the following important class of matrices.

**Definition 8** We will say that a matrix $A \in \mathscr{M}^n \subset \mathbb{R}_{max}^{n\times n}$ if:

$$a_{i+1,j} \geq a_{ij},\ i \leq n-1,\ j \leq n, \tag{6}$$

$$a_{ij} \geq a_{i,j+1},\ i \leq n,\ j \leq n-1, \tag{7}$$

$$\xi_{1j}(A) \geq \ldots \geq \xi_{nj}(A),\ j \leq n, \tag{8}$$

$$a_{ij} > -\infty,\ j \leq i+1,\ i \leq n,\ j \leq n. \tag{9}$$

The main result characterizing the Flow Shop Model is expressed in the following theorem.

**Theorem 1** *Given* $(\mathbf{c},\ \boldsymbol{\tau})(j)$*, for some* $j \in \mathscr{J}$*. If we construct* $A(j)$ *according to the algorithm in* Fig. 5*,* $A(j) \in \mathscr{M}^n$*.*

The proof to this theorem is in Appendix.

## 4 Problem formulation

Let $\mathscr{A}$ be a set of $m$ distinct matrices in $\mathscr{M}^n$ indexed by the set $\mathscr{J} = \{1, \ldots, m\}$. We consider a class of input quantized systems of the form

$$\begin{aligned} \mathbf{x}(k+1) &= A(p(k)) \otimes \mathbf{x}(k) \\ y(k) &= \|A(p(k)) \otimes \mathbf{x}(k)\|_{1_{max}} \oslash \|\mathbf{x}(k)\|_{1_{max}}, \end{aligned} \tag{10}$$

where $\mathbf{x}(k) \in \mathbb{R}^n$, $y(k) \in \mathbb{R}$, $p(k) \in \mathscr{J}$, and $A(p(k)) \in \mathscr{A}$. Batch flow shops can be represented as systems of this form. Thus, these dynamics represent a generalization of batch flow shops to the set $\mathscr{M}^n$.

Given a vector $\mathbf{q} \in \mathbb{N}^m$, such that $Q = \sum_{i=1}^m q_i$, we say that a sequence $p = (p(0), \ldots, p(|\mathbf{q}|_1 - 1))$, with $p(i) \in \mathscr{U}$ is *admissible* if

$$\sum_{i=0}^{Q-1} I_j(p(i)) = q_j \ \forall\ 1 \leq j \leq m\,,$$

where $I_j(k)$ is the indicator function:

$$I_j(k) = \begin{cases} 1 & \text{if } j = k, \\ 0 & \text{otherwise.} \end{cases}$$

Characterizing admissible inputs to the system then leads to the following problem:

$$\begin{aligned} \min_{p \text{ admissible}} \ & \sum_{k=0}^{Q-1} y(k) \\ \text{subject to } & \mathbf{x}(k+1) = A(p(k)) \otimes \mathbf{x}(k) \\ & y(k) = \|A(p(k)) \otimes \mathbf{x}(k)\|_{1_{max}} \oslash \|\mathbf{x}(k)\|_{1_{max}}. \end{aligned} \tag{11}$$

When each $A \in \mathscr{A}$ represents the recipe in a batch flow shop, and $\mathbf{q}$ is interpreted as a fixed quota, this problem is equivalent to the makespan minimization problem with respect to a quota. We will now show that this problem is $\mathscr{NP}$-complete, similar to what is shown in Su and Woeginger (2011).

**Proposition 1** *The problem given in* Eq. 11 *is* $\mathscr{NP}$*-complete.*

*Proof* First we must show that our problem is in $\mathscr{NP}$. This is trivial since an admissible sequence can be easily constructed, and checking the solution is done by calculating $\mathbf{x}_Q$ and then $\|\mathbf{x}_Q\|_{1_{max}} \oslash \|\mathbf{x}_0\|_{1_{max}}$. These can all be done in polynomial time.

To show our problem is $\mathscr{NP}$-complete, we will reduce $F_3|block|C_{max}$, which is the optimal sequencing of the 3-machine flowshop with blocking with respect to makespan, to our problem. This problem is shown to be $\mathscr{NP}$-complete in Hall and Sriskandarajah (1996). This problem can be represented as a 3 machine batch flowshop with machine capacities all equal. The algorithm in Fig. 5 is a polynomial time algorithm that transforms a batch flowshop to a set of matrices in $\mathscr{M}^n$. Thus $F_3|block|C_{max}$ is reducible to our problem in polynomial time. □

Because problem (11) is $\mathscr{NP}$-complete it is already hard to solve. What makes it even more intractable is the dependence of cost function $y_k$ on all the previous inputs. Unlike problems like Traveling Salesperson Problem, where cost for visiting a city is known a-priori, in this problem, the cost of a job is unknown until all the previous inputs are decided. In this paper, we will give an approximation method for the cost function, so that it can be calculated with the knowledge of only a few previous inputs. Then we formulate an integer programming problem to compute a suboptimal solution. We will also show that the error of this solution is bounded. First, we go over some of the properties of our system which will motivate the approximation method and will be used to compute the error bounds.

## 5 Max-plus systems generated by $\mathscr{M}^n$

### 5.1 Properties of $\mathscr{M}^n$

Because of the structure of matrices in $\mathscr{M}^n$, it has many useful properties. In this subsection we will show that the set $\mathscr{M}^n$ is closed under max-plus multiplication. We will also prove some Lemmas that will be useful in the next section.

**Theorem 2** *Suppose* $A, B \in \mathscr{M}^n$. *Then* $A \otimes B \in \mathscr{M}^n$.

*Proof* Let $A, B \in \mathscr{M}^n$ be given. We will write $C = A \otimes B$. To show that $C \in \mathscr{M}^n$, we must show that all four equations in Definition 8 hold. We will show each individually.

Equation 6, $c_{ij} \leq c_{i+1,j}$: Let $i, j \leq n$ be given. We will pick $\kappa$ such that $c_{ij} = a_{i\kappa} \otimes b_{\kappa j}$. Then,

$$\begin{aligned} c_{ij} \oslash c_{i+1,j} &\leq a_{i\kappa} \otimes b_{\kappa j} \oslash (a_{i+1,\kappa} \otimes b_{\kappa j}) \\ &= a_{i\kappa} \oslash a_{i+1,\kappa} \\ &\leq e. \end{aligned}$$

Equation 7, $c_{ij} \geq c_{i,j+1}$: Let $i, j \leq n$ be given. We will pick $\kappa$ such that $c_{ij+1} = a_{i\kappa} \otimes b_{\kappa,j+1}$. Then,

$$\begin{aligned} c_{ij} \oslash c_{i,j+1} &\geq a_{i\kappa} \otimes b_{\kappa j} \oslash (a_{i\kappa} \otimes b_{\kappa,j+1}) \\ &= b_{\kappa j} \oslash b_{\kappa,j+1} \\ &\geq e. \end{aligned}$$

Equation 8, $c_{ij} \oslash c_{i,j+1} \geq c_{i+1,j} \oslash c_{i+1,j+1}$: Let $i, j < n$ be given. We will pick $\kappa, l, r, s$ such that

$$c_{ij} = a_{i\kappa} \otimes b_{\kappa j} \tag{12}$$
$$c_{i,j+1} = a_{il} \otimes b_{l,j+1} \tag{13}$$
$$c_{i+1,j} = a_{i+1,r} \otimes b_{rj} \tag{14}$$
$$c_{i+1,j+1} = a_{i+1,s} \otimes b_{s,j+1}. \tag{15}$$

From Eqs. 12, 15 we can derive the following inequalities

$$a_{i\kappa} \oslash a_{il} \geq b_{lj} \oslash b_{\kappa j} \tag{16}$$
$$a_{i+1,s} \oslash a_{i+1,r} \geq b_{r,j+1} \oslash b_{s,j+1} \tag{17}$$
$$b_{s,j+1} \oslash b_{l,j+1} \geq a_{i+1,l} \oslash a_{i+1,s} \tag{18}$$
$$b_{\kappa j} \oslash b_{rj} \geq a_{ir} \oslash a_{i\kappa}. \tag{19}$$

Now consider

$$\begin{aligned} \omega =& c_{ij} \oslash c_{i,j+1} \oslash (c_{i+1,j} \oslash c_{i+1,j+1}) \\ =& a_{i\kappa} \otimes b_{\kappa j} \oslash a_{il} \oslash b_{l,j+1} \\ & \oslash a_{i+1,r} \oslash b_{rj} \otimes a_{i+1,s} \otimes b_{s,j+1} \\ =& (a_{i\kappa} \oslash a_{il}) \otimes (a_{i+1,s} \oslash a_{i+1,r}) \\ & \otimes (b_{s,j+1} \oslash b_{l,j+1}) \otimes (b_{\kappa j} \oslash b_{rj}). \end{aligned} \tag{20}$$

From this equation we will consider two cases.

Suppose $l \leq r$. Then we write

$$\begin{aligned} \omega \geq& b_{lj} \oslash b_{\kappa j} \otimes b_{\kappa j} \oslash b_{l,j+1} && (21) \\ & \otimes b_{r,j+1} \oslash b_{s,j+1} \oslash b_{rj} \otimes b_{s,j+1} \\ =& b_{lj} \oslash b_{l,j+1} \oslash (b_{rj} \oslash b_{r,j+1}) && (22) \\ \geq& e. && (23) \end{aligned}$$

Where we obtain Eq. 21 by substituting Eqs. 16 and 17 into Eq. 20, Eq. 22 by canceling and rearranging terms, and Eq. 23 by Eq. 8.

Suppose $l > r$. Then we write

$$\begin{aligned}\omega \geq & a_{i\kappa} \oslash a_{i+1,r} \otimes a_{ir} \oslash a_{i\kappa} \\ & \otimes a_{i+1,s} \oslash a_{il} \otimes a_{i+1,l} \oslash a_{i+1,s} \\ = & a_{ir} \oslash a_{il} \oslash (a_{i+1,r} \oslash a_{i+1,l}) \\ \geq & e.\end{aligned} \tag{24}$$

(25)

(26)

Where we obtain Eq. 24 by substituting Eqs. 18 and 19 into Eq. 20, Eq. 25 by canceling and rearranging terms, and Eq. 26 by Eq. 8.

Equation 9: Let $i, j$ such that $j \leq i+1$, $i, j \leq n$. Then

$$\begin{aligned}c_{ij} &= \bigoplus_{\kappa=1}^{n} a_{i\kappa} \otimes b_{\kappa j} \\ &\geq a_{ii} \otimes b_{ij} \\ &> -\infty.\end{aligned}$$

□

To simplify the notation in the following results, we will use this definition.

**Definition 9** For some $A \in \mathscr{M}^n$, we define

$$\begin{aligned}Z_i(A) &= a_{in} \oslash a_{i-1,n} \\ z_i(A) &= a_{i1} \oslash a_{i-1,1}.\end{aligned}$$

The following Lemma, taken from Weyerman and Warnick (2007) gives us a range on the "spread" that can occur between the elements of the state vector after applying any input.

**Lemma 3** (Weyerman and Warnick 2007) *For some $A \in \mathscr{M}^n$, for any $\mathbf{x} \in \mathbb{R}^n_{\max}$, if we let $\mathbf{y} = A \otimes \mathbf{x}$, then*

$$z_i(A) \leq y_i \oslash y_{i-1} \leq Z_i(A).$$

These results equips us to make some statements related to the input-output properties of these max-plus operators.

**Proposition 2** *Suppose $A, B \in \mathscr{M}^n$ and $\mathbf{x}(\ell) = \begin{bmatrix} \epsilon & \dots & \epsilon & e \end{bmatrix}^T$. Then*

$$\mathbf{x}(\ell) = \arg\min_{\mathbf{x} \neq \epsilon} \left( \|A \otimes \mathbf{x}\|_{1_{max}} \oslash \|\mathbf{x}\|_{1_{max}} \right)$$

*and*

$$\mathbf{x}(\ell) = \arg\min_{\mathbf{x} \neq \epsilon} \left( \|A \otimes B \otimes \mathbf{x}\|_{1_{max}} \oslash \|B \otimes \mathbf{x}\|_{1_{max}} \right). \tag{27}$$

Note that $\mathbf{x}(\ell)$ is the best possible state for the system to complete its subsequent jobs as quickly as possible.

*Proof* Let $A \in \mathscr{M}^n$ be given. By Lemma 2 and Definition 8, we know that

$$\begin{aligned}\min_{\mathbf{x} \neq \epsilon} \left( \|A \otimes \mathbf{x}\|_{1_{max}} \oslash \|\mathbf{x}\|_{1_{max}} \right) &= \min_i \left[ \mathbf{e}^T \otimes A \right]_i \\ &= a_{nn}.\end{aligned}$$

Consider $\mathbf{x} = \begin{bmatrix} \epsilon & \dots & \epsilon & e \end{bmatrix}^T$, then $\|\mathbf{x}\|_{1_{max}} = e$, and

$$\begin{aligned} \|A \otimes \mathbf{x}\|_{1_{max}} &= \left\| \begin{bmatrix} a_{n1} & \dots & a_{nn} \end{bmatrix} \right\|_{1_{max}} \\ &= a_{nn}. \end{aligned}$$

To show Eq. 27, we will also suppose that $B \in \mathscr{M}^n$. Let $\tilde{\mathbf{y}} = B \otimes \tilde{\mathbf{x}}$ with $\tilde{\mathbf{x}} = \begin{bmatrix} \epsilon & \cdots & e \oslash b_{nn} \end{bmatrix}^T$ and suppose that there is some $\bar{\mathbf{y}} = B \otimes \bar{\mathbf{x}}$ with $\|\bar{\mathbf{y}}\|_{1_{max}} = e$ such that $\|A \otimes \bar{\mathbf{y}}\|_{1_{max}} < \|A \otimes \tilde{\mathbf{y}}\|_{1_{max}}$.

We will pick $j, \kappa$ such that $\|A \otimes \tilde{\mathbf{y}}\|_{1_{max}} = a_{nj} \otimes \tilde{y}_j$ and $\|A \otimes \bar{\mathbf{y}}\|_{1_{max}} = a_{n\kappa} \otimes \bar{y}_\kappa$. Then we have the following inequalities

$$\begin{aligned} a_{nj} \otimes \tilde{y}_j &> a_{n\kappa} \otimes \bar{y}_\kappa \\ a_{n\kappa} \otimes \bar{y}_\kappa &\geq a_{nj} \otimes \bar{y}_j. \end{aligned}$$

These can be combined to get

$$\bar{y}_j < \tilde{y}_j.$$

But, by Lemma 3,

$$\begin{aligned} \bar{y}_n &\leq \bar{y}_j \otimes \bigotimes_{i=j+1}^{n} Z_i(B) \\ &< \tilde{y}_j \otimes \bigotimes_{i=j+1}^{n} Z_i(B) \\ &= e. \end{aligned}$$

Which contradicts the statement that $\|\bar{\mathbf{y}}\|_{1_{max}} = e$, so $\tilde{\mathbf{y}}$ and hence $\tilde{\mathbf{x}}$ achieves the minimum. It is easy to see that $\mathbf{x} = \tilde{\mathbf{x}} \otimes b_{nn}$ also achieves the minimum. □

### 5.2 Fixed input stability

Now, we will show that the system (10), which represents a batch flow shop, is stable under the same input or the same sequence of inputs. To show this result we will need several preliminary lemmas.

Let $\delta_i$ be the difference between the rows of the first column as follows:

$$\delta_i(A) = a_{i+1,1} - a_{i1}$$

We can write any matrix, $A \in \mathscr{M}^n$, in terms of $a_{11}$, the row difference $\xi_{ij}$ and the column difference $\delta_i$'s as

$$a_{ij} = a_{11} - \sum_{l=1}^{j-1} \xi_{il} + \sum_{\kappa=1}^{i-1} \delta_\kappa. \tag{28}$$

We note that a matrix, $A \in \mathbb{R}_{\max}^{n \times n}$ can be used to represent a directed graph with a set of $n$ nodes or vertices, $\mathscr{N}(A) = \{1, \dots, n\}$, and a set of edges or arcs, $\mathscr{D}(A)$, which are ordered pairs of vertices. We define $a_{ij}$ as the edge weight from node $j$ to node $i$. If $a_{ij}$ is greater than $-\infty$, then $(v_j, v_i) \in \mathscr{D}(A)$, meaning there is an edge from node $j$ to node $i$, otherwise $(v_j, v_i) \notin \mathscr{D}(A)$. For a matrix $A \in \mathbb{R}_{\max}^{n \times n}$, we will denote $\mathscr{G}(A) =: \{\mathscr{N}(A), \mathscr{D}(A)\}$ to mean the graph representation of the matrix $A$.

Given two distinct vertices, $v_i, v_j \in \mathcal{N}(A)$, we will define the function $h(v_i, v_j) : \mathcal{N}(A) \times \mathcal{N}(A) \to \mathbb{R}_{\max}$ as

$$h(v_i, v_j) = \begin{cases} -\sum_{\kappa=v_i}^{v_j-1} \xi_{v_j,\kappa} & v_i < v_j \\ \sum_{\kappa=v_j}^{v_i-1} \xi_{v_j,\kappa} & v_i > v_j \end{cases}.$$

For a circuit in a graph, $\mathcal{V} = \{v_1, \ldots, v_l\}$, with $|\mathcal{V}| =: l$, we will use the convention that $v_{l+1} = v_1$. The average circuit weight of a circuit $\mathcal{V}$, $w(\mathcal{V})$, is defined as

$$w(\mathcal{V}) =: \frac{\sum_{v_i \in \mathcal{V}} a_{v_{i+1},v_i}}{|\mathcal{V}|}$$

**Lemma 4** *Given a circuit, $\mathcal{V} = \{v_1, v_2, \ldots, v_l\}$ in $\mathcal{G}(A)$ with $A \in \mathcal{M}^n$, $\sum_{v_i \in \mathcal{V}} h(v_i, v_{i+1}) \leq 0$.*

*Proof* We will let $b$ correspond to a vertex in $\mathcal{N}(A)$ and construct a sum of over the vertices in $\mathcal{V}$ with respect to $b$. We will then show that the sum with respect to each $b$, $0 < b < n$ is less than or equal to zero, and that the sum over all $0 < b < n$ is the sum in question.

Let $0 < b < n$ be given. To construct our sum with respect to $b$, we will first consider the arcs that "pass" $b$ in the positive direction. That is, if there is some $i$ such that $v_i \leq b < v_{i+1}$ we will add $\xi_{v_{i+1},b}$. Let $I_b$ be the set of all such $i$'s. Now, consider the arcs that "pass" $b$ in the negative direction. That is, if there is some $j$ such that $v_{j+1} \leq b < v_j$ we will add $-\xi_{v_{j+1},b}$. Let $J_b$ be the set of all such $j$'s.

Note that since $\mathcal{V}$ is a circuit we will have $v_i \leq b < v_{i+1}$ for some $i$ if and only if there is some $j \neq i$ such that $v_{j+1} \leq b < v_j$. This is true because $\mathcal{V}$ is a circuit, each time we "pass" node $b$, we must "pass" it going the other way to eventually return to the start node. Hence, $I_b$ and $J_b$ have the same number of elements.

Taking the sum over $I_b$ and $J_b$, we get $\sum_{i \in I_b} \xi_{v_{i+1},b} - \sum_{j \in J_b} \xi_{v_{j+1},b}$. By definition, for every $j \in J_b$, $v_{j+1} \leq b$ and for every $i \in I_b$, $b < v_{i+1}$; thus, for every $i \in I_b$, $j \in J_b$, $v_{j+1} < v_{i+1}$. From this and by Definition 8, $\xi_{v_{j+1},b} \geq \xi_{v_{i+1},b}$ for every $i \in I_b$ and $j \in J_b$. Hence, $\sum_{i \in I_b} \xi_{v_{i+1},b} - \sum_{j \in J_b} \xi_{v_{j+1},b} \leq 0$.

We will generalize the notions of $I_b$ and $J_b$ to $\mathcal{I} = \{i | v_i < v_{i+1}; v_i, v_{i+1} \in \mathcal{V}\}$ and $\mathcal{J} = \{j | v_{j+1} < v_j; v_j, v_{j+1} \in \mathcal{V}\}$. Then

$$\begin{aligned} \sum_{v_i \in \mathcal{V}} h(v_i, v_{i+1}) &= \sum_{i \in \mathcal{I}} \sum_{\kappa=v_i}^{v_{i+1}-1} \xi_{v_{i+1},\kappa} - \sum_{j \in \mathcal{J}} \sum_{\kappa=v_{j+1}}^{v_j-1} \xi_{v_{j+1},\kappa} \\ &= \sum_{b=1}^{n-1} \left( \sum_{i \in I_b} \xi_{v_{i+1},b} - \sum_{j \in J_b} \xi_{v_{j+1},b} \right) \\ &\leq 0. \end{aligned}$$

□

**Lemma 5** *Let $a_{rr}$ be the maximum diagonal element in a matrix, $A \in \mathscr{M}^n$. Then*

$$\sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} \delta_i \leq 0$$

*for any $j \leq n$ with equality if and only if $a_{rr} = a_{jj}$.*

*Proof* Let $j \leq n$ be given. We can write

$$a_{jj} = a_{rr} + \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} \delta_i.$$

Because $a_{jj}$ is on the diagonal, $a_{jj} \leq a_{rr}$, so

$$\begin{aligned} \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{j-1} \xi_{ji} + \sum_{i=1}^{j-1} \delta_i &= a_{jj} - a_{rr} \\ &\leq 0 \end{aligned}$$

and equality is achieved if and only if $a_{jj} = a_{rr}$. □

**Lemma 6** *Let $\mathscr{V}$ be a circuit in $\mathscr{G}(A)$ with $A \in \mathscr{M}^n$, where $a_{rr}$ is the maximum diagonal element of $A$. Then the average circuit weight $w(\mathscr{V}) \leq a_{rr}$ with equality if and only if $a_{jj} = a_{rr} \forall j \in \mathscr{V}$.*

*Proof* Let $l = |\mathscr{V}|$. We will consider $lw(\mathscr{V})$. Then

$$lw(\mathscr{V}) = a_{v_2 v_1} + a_{v_3 v_2} + \ldots + a_{v_l v_1} \tag{29}$$

$$= a_{11} - \sum_{i=1}^{v_1-1} \xi_{v_2 i} + \sum_{i=1}^{v_1-1} \delta_i \tag{30}$$

$$+ a_{11} - \sum_{i=1}^{v_2-1} \xi_{v_3 i} + \sum_{i=1}^{v_2-1} \delta_i + \ldots$$

$$+ a_{11} - \sum_{i=1}^{v_l-1} \xi_{v_1 i} + \sum_{i=1}^{v_l-1} \delta_i$$

$$= l a_{rr} \tag{31}$$

$$+ \left( \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{v_l-1} \xi_{v_1 i} + \sum_{i=1}^{v_1-1} \delta_i \right)$$

$$+ \left( \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{v_1-1} \xi_{v_2 i} + \sum_{i=1}^{v_2-1} \delta_i \right)$$

$$+ \ldots$$

$$+ \left( \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{v_{l-1}-1} \xi_{v_l i} + \sum_{i=1}^{v_l-1} \delta_i \right)$$

Where Eq. 29 follows by definition of $w(\mathscr{V})$, Eq. 30 follows by substitution as in Eq. 28, and Eq. 31 follows by substitution of $a_{11} = a_{rr} + \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i$ from Eq. 28.

We can decompose $\sum_{i=1}^{v_j-1} \xi_{v_{j+1}i}$ as $\sum_{i=1}^{v_{j+1}-1} \xi_{v_{j+1}i} + h(v_j, v_{j+1})$. Thus by Lemma 5, each

$$\begin{aligned}
&\sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{v_j-1} \xi_{v_{j+1}i} + \sum_{i=1}^{v_{j+1}-1} \delta_i \\
&\quad = \sum_{i=1}^{r-1} \xi_{ri} - \sum_{i=1}^{r-1} \delta_i - \sum_{i=1}^{v_{j+1}-1} \xi_{v_{j+1}i} + \sum_{i=1}^{v_{j+1}-1} \delta_i \\
&\qquad + h(v_j, v_{j+1}) \\
&\quad \leq h(v_j, v_{j+1}).
\end{aligned}$$

With equality when $a_{jj} = a_{rr}$.

Thus

$$\begin{aligned}
l w(\mathscr{V}) &\leq l a_{rr} + \sum_{v_j \in \mathscr{V}} h(v_j, v_{j+1}) && (32) \\
&\leq l a_{rr} && (33) \\
w(\mathscr{V}) &\leq a_{rr}.
\end{aligned}$$

Here (32) follows from Lemma 4. We get equality in Eqs. 32, 33 if and only if $a_{jj} = a_{rr}\ \forall j \in \mathscr{V}$. □

**Lemma 7** *Let $A \in \mathscr{M}^n$. Then every vertex in the critical graph of $A$ has a self loop.*

*Proof* Let $v$ be a vertex in the critical graph of $A$. This means that $v$ is in a critical circuit of $A$ which we will call $\mathscr{V}$. By Lemma 6, it must be that $w(\mathscr{V}) = a_{rr}$ because vertex $r$ has a self loop with average weight $a_{rr}$ which must be in the critical graph. Furthermore, since $v \in \mathscr{V}$, $a_{vv} = a_{rr}$, so the self loop on $v$ must also be in the critical circuit of $A$. □

**Theorem 3** *Let $A \in \mathscr{M}^n$. Then $A$ has cyclicity one.*

*Proof* By Lemma 7, every vertex in the critical graph of $A$ has a self loop. In Heidergott et al. (2006) the cyclicity of a matrix is defined to be the cyclicity of the critical graph of that matrix. We will call $G$ the critical graph of $A$.

Suppose that $G$ is strongly connected. The cyclicity of $A$ is the greatest common divisor of the lengths of all elementary circuits in $G$. Because every node in $G$ has a self loop, the elementary circuits of those self-loops have length one, thus the cyclicity of $G$ must be one.

Suppose that $G$ is not strongly connected. Then the cyclicity of $G$ (and thus $A$) is the least common multiple of the cyclicities of all maximal strongly connected subgraphs (m.s.c.s.'s) of $G$. Again, since each node in each m.s.c.s. of $G$ has a self-loop, the cyclicity of each m.s.c.s. is one and thus the cyclicity of $G$ is one. □

**Lemma 8** *Let $A \in \mathscr{M}^n$. Then $A$ is irreducible.*

*Proof* Let $A \in \mathscr{M}^n$ be given. Recall that a matrix is irreducible if $\mathscr{G}(A)$ is strongly connected. We will enumerate the vertices of $\mathscr{G}(A)$ as $\{v_1, \ldots, v_n\}$. It is easy to see that for $j \leq i$, $v_i$ is reachable from $v_j$ since $a_{ij} > -\infty$ by Eq. 9. Suppose $j > i$, then we

see by Eq. 9 that $v_{j-1}$ is reachable from $v_j$, and we can reach $v_i$ by traversing the edges $(v_j, v_{j-1}), (v_{j-1}, v_{j-2}), \ldots, (v_{i+1}, v_i)$. Therefore, $\mathcal{G}(A)$ is strongly connected and $A$ is irreducible. □

From Cohen et al. (1985), Ahmadi et al. (1992) we get the following theorem which we will need in our proof of stability.

**Theorem 4** (Cohen et al. 1985, Ahmadi et al. 1992) *Let $A \in \mathbb{R}_{\max}^{n\times n}$ be an irreducible matrix with eigenvalue $\lambda$ and cyclicity $\sigma = \sigma(A)$. Then there is an $N$ such that*

$$A^{\otimes(\kappa+\sigma)} = \lambda^{\otimes\sigma} \otimes A^{\otimes\kappa}$$

*for all $\kappa \geq N$.*

Now we are ready to prove the main theorem of this section.

**Theorem 5** *Let $A \in \mathcal{M}^n$. Then the linear autonomous system*

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k)$$

*is stable in the sense of Definition 3.*

*Proof* Let $A \in \mathcal{M}^n$ with eigenvalue $\lambda$. By Lemma 8 and Theorems 3 and 4, we know that there is some $N$ such that

$$A^{\otimes(l+1)} = \lambda \otimes A^{\otimes l}$$

for all $l \geq N$. Let $\mathbf{x} \in \mathbb{R}_{\max}^n$, $l \geq N$ be given. Let $i \leq n$ be given. Consider $A^{\otimes(l+1)}\mathbf{x} = \lambda \otimes A^{\otimes l} \otimes \mathbf{x}$. From this we see that after $l$ steps, $A^{\otimes l} \otimes \mathbf{x}$ is an eigenvector of $A$, therefore we see that for all $k \geq 1$, $[A^{\otimes(k+l)} \otimes \mathbf{x}]_i \oslash [A^{\otimes(k+l)} \otimes \mathbf{x}]_1$ is a constant value. □

**Corollary 1** *Let $A(i) \in \mathcal{M}^n$, $i = 1, 2, ..., t$. Then the linear autonomous system*

$$\mathbf{x}(k+1) = A(1) \otimes ... \otimes A(t) \otimes \mathbf{x}(k)$$

*is stable in the sense of Definition 3.*

*Proof* The proof follows directly from the fact that $\mathcal{M}^n$ is closed under max-plus multiplication and Theorem 5. □

## 6 Suboptimal scheduling with bounds

### 6.1 Model approximation

Since any system in $\mathcal{M}^n$ is stable, clearly any system in our set $\mathcal{A} \subset \mathcal{M}^n$ is stable. So, if the input, $p$, is constant, the effect of the initial condition dies away as the system stabilizes. This motivates us to pose an approximation to the system that assumes only the most recent inputs effect the current state. If we consider that only the last $t$ inputs have an effect and hence only those inputs are used in the computation of $y(k)$, the approximation is called a *$t$-step approximation*. Note that the choice of a good approximation size for a particular kind of system is a subject of ongoing research, although the general idea would be to choose $t$ as large as possible, such that the computational effort required to optimally solve the resulting (smaller) sequencing problem is within the bounds of available computational resources.

To simplify notation, we will denote the sequence of inputs from time $i$ to time $j$ as $P(i, j) = (p(i), p(i+1), \ldots, p(j-1), p(j))$ with $i \leq j$. Using this notation, we will also denote

$$A(P(i, j)) = A(p(j)) \otimes \cdots \otimes A(p(i))$$

and note that, by Theorem 2, $A(P(i, j)) \in \mathscr{M}$. For a sequence $P(0, k) = (p(0), \ldots, p(k))$, we will write

$$\begin{aligned} \mathbf{x}(k+1) &= A(P(0,k)) \otimes \mathbf{x}(0) \\ y(P(0,k)) &= \mathbf{x}(k+1) \oslash \mathbf{x}(0) \\ &= \sum_{i=0}^{k} y(i). \end{aligned}$$

If the system is driven by a sequence $P(0, k-1) = (p(0), \ldots, p(k-1))$, then we have as a solution to Eq. 10

$$\mathbf{x}(k) = A(P(0, k-1)) \otimes \mathbf{x}(0).$$

We want to build our approximation such that we lower bound the actual output, so we approximate the current state using the *t-step approximation* for the subsequence $P(k-t, k-1)$.

**Definition 10** Given a system as in Eq. 10 and a sequence of inputs, $P(0, k) = (p(0), \ldots, p(k))$, we define the $t$-step approximation of $(x, y)$ to be

$$\begin{aligned} \hat{\mathbf{x}}^{\mathbf{t}}(k) &= \begin{cases} A(P(k-t, k-1)) \otimes \mathbf{x}(\ell), & \text{if } k > t, \\ A(P(0, k-1)) \otimes \mathbf{x}(0), & \text{otherwise} \end{cases} \\ \hat{y}^t(k) &= \|A(p(k)) \otimes \hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}} \oslash \|\hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}} \end{aligned}$$

with $\mathbf{x}(\ell) = \begin{bmatrix} \epsilon & \ldots & \epsilon & e \end{bmatrix}^T$.

Here, $\hat{y}^p(k)$ approximates the cost of completing the job $p(k)$ after completing the sequence of jobs $P(k-t, k-1)$.

This approximation leads to an approximation of the problem in Eq. 11:

$$\begin{aligned} \min_{P \text{ admissible}} & \sum_{k=0}^{Q-1} \hat{y}^t(k) \\ \text{subject to } \hat{\mathbf{x}}^{\mathbf{t}}(k) &= \begin{cases} A(P(k-t, k-1)) \otimes \mathbf{x}(\ell), & \text{if } k > t, \\ A(P(0, k-1)) \otimes \mathbf{x}(0), & \text{otherwise} \end{cases} \\ \hat{y}^t(k) &= \|A(p(k)) \otimes \hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}} \oslash \|\hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}}. \end{aligned} \tag{34}$$

This problem is easier to solve than Eq. 11 because we reduce the amount of computation it takes to obtain the transition cost, $y(k)$. Finding the best schedule using any algorithm requires the calculation of the transition cost. When the approximation is not used, calculation of a $y(k)$ requires the knowledge of the state $x(k)$, which depends on the inputs $P(0, k-1)$. But using the approximate, the current state $\hat{\mathbf{x}}^{\mathbf{t}}(k)$ only depends on $P(k-t, k-1)$. For example, let us suppose we have a factory that operates on three different job types $p_1$, $p_2$, and $p_3$. Also suppose that the factory gets a sequence of jobs, say $p_1$, $p_2$, $p_3$, $p_1$, $p_3$. The actual cost of producing the last $p_3$ is given by $\|A(p_3) \otimes \mathbf{x}(4)\|_{1_{max}} \oslash \|\mathbf{x}(4)\|_{1_{max}}$, where $\mathbf{x}(4) = A_{p_1} \otimes A(p_3) \otimes A(p_2) \otimes A(p_1) \otimes \mathbf{x}_0$. If a 2-step approximation is used the cost is given by $\|A(p_3) \otimes \hat{\mathbf{x}}^{\mathbf{2}}(4)\|_{1_{max}} \oslash \|\hat{\mathbf{x}}^{\mathbf{2}}(4)\|_{1_{max}}$, where $\hat{\mathbf{x}}^{\mathbf{2}}(4) = A(p_1) \otimes A(p_3) \otimes \mathbf{x}(\ell)$.

While using a $t$-step approximation, additional speedup can be gained by calculating and storing all the possible values of $\hat{y}^t(k)$ before solving the optimization problem and reusing them during the solution. This is especially helpful if the size of the quota is much larger than the number of distinct job types, which is common in practical situations.

### 6.2 Scheduling method

To solve the scheduling problem for the $t$-step approximation, we will formulate an integer program similar to that of the traveling salesperson problem. Let $\mathscr{P}(t)$ be the set of all $t$ long sequences of job types, $\mathscr{P}(t) = \{(p(1), \ldots, p(t)) | p(i) \in (\mathscr{J} \cup 0)\}$, where 0 is a new job type corresponding to allowing the flow shop to complete processing of all the current jobs and

$$A(0) = \begin{bmatrix} e & \ldots & e \\ \vdots & \ddots & \vdots \\ e & \ldots & e \end{bmatrix}.$$

Consider a graph with a vertex corresponding to each sequence in $\mathscr{P}_t$ and directed edges from vertex representing the sequences $(p(i), ..., p(i+t-1))$ to $(p(i+1), ..., p(i+t-1), p(i+t))$. The edge weight is given by the approximate cost of completing the job $p(i+t)$ after completing the sequence of jobs $(p(i), ..., p(i+t-1))$. Hence each node corresponds to the completion of a job and the weights are the different values of $\hat{y}^t(k)$. Now, the problem of finding the sequence of jobs that minimizes the makespan while fulfilling the quota is the same as finding the shortest tour in the graph that visits the nodes, possibly multiple times, to fulfill the quota.

This graph can be represented by a weight matrix $C \in \mathbb{R}^{(m+1)^t \times (m+1)}$ whose rows are indexed by different sequences in $\mathscr{P}_t$ and columns are indexed by the job types. Let decision variable $W \in \mathbb{N}^{(m+1)^t \times (m+1)}$ indicate the number of times we traverse each edge. Because we are minimizing makespan, we have as the objective function:

$$\min_W \left( \sum_{j=1}^{m+1} \sum_{i=1}^{(m+1)^t} c_{ij} w_{ij} \right)$$

We construct the first constraint to enforce traversal of the graph as tours, i.e. for every node the number of times the incoming edges are traversed is equal to the number of times the outgoing nodes are traversed:

$$\sum_{i=1}^{m+1} w_{pj} - \sum_{K} w_{Kl} = 0;$$
$$p = (p(1), ..., p(t)) \in \mathscr{P}(t), l = p(t),$$
$$K = \{(x, p(1), p(2), ..., p_{t-1}) | x \in (\mathscr{J} \cup 0)\}$$

The second constraint requires that the quota is met. We will set the quota of job 0 to be 1.

$$\sum_{i=1}^{(m+1)^t} w_{ij} = q_j; \quad j = 1, \ldots, m+1. \tag{35}$$

Combining these, we get the integer programming formulation

$$\begin{aligned}
\min_{W} \quad & \left( \sum_{j=1}^{m+1} \sum_{i=1}^{(m+1)^t} c_{ij} w_{ij} \right) \\
\text{subject to: } & \sum_{i=1}^{m+1} w_{pj} - \sum_{K} w_{Kl} = 0; \\
& \qquad p = (p(1), ..., p(t)) \in \mathscr{P}(t), l = p(t), \\
& \qquad K = \{(x, p(1), p(2), ..., p_{t-1}) | x \in (\mathscr{J} \cup 0)\} \\
& \sum_{i=1}^{(m+1)^t} w_{ij} = q_j; \quad j = 1, \ldots, m+1 \\
& w_{ij} \in \{0, 1, \ldots\}
\end{aligned}$$

The solution of this problem may lead to multiple sub-tours. We require a single tour. In order to achieve a single tour, we employ a strategy presented in Vanderbei (2001). The problem is iteratively solved adding a constraint each iteration to break the smallest sub-tour, $\varsigma$, of length $\varsigma_{length}$:

$$\sum_{i,j \in \varsigma} w_{ij} < \varsigma_{length} \tag{36}$$

Once there is a single tour, the matrix $W$ will represent a directed graph and will specify the number of times each arc is to be traversed. The tour specified by $W$ is the schedule.

### 6.3 Error bounds

To compute the bounds on the error generated by the solution of the approximated problem, we will first show that our approximation method lower bounds the actual output. We use $\mathbf{x}(\ell) = \begin{bmatrix} \epsilon & \ldots & \epsilon & e \end{bmatrix}^{\mathbf{T}}$ in order to arrive at the following proposition.

**Proposition 3** *Given a system as in Eq. 10 and a sequence $P(0, k)$, the output of the $t$-step approximation $\hat{y}^t(k)$ gives a lower bound of the actual output $y(k)$, i.e. it satisfies*

$$\hat{y}^t(k) \leq y(k)$$

*for all $k > t$, and*

$$\hat{y}^t(k) = y(k)$$

*for all $k \leq t$.*

*Proof* Let a system as in Eq. 10 and a sequence $P(0, k)$ be given. By Proposition 2 and Theorem 2

$$\begin{aligned}
\hat{y}^t(k) =& \|A(p(k)) \otimes \hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}} \oslash \|\hat{\mathbf{x}}^{\mathbf{t}}(k)\|_{1_{max}} \\
=& \|A(p(k)) \otimes A(P(k-t, k-1)) \otimes \mathbf{x}(\ell)\|_{1_{max}} \\
& \oslash \|A(P(k-t, k-1)) \otimes \mathbf{x}(\ell)\|_{1_{max}} \\
=& \min_{\mathbf{x}} \|A(p(k)) \otimes A(P(k-t, k-1)) \otimes \mathbf{x}\|_{1_{max}} \\
& \oslash \|A(P(k-t, k-1)) \otimes \mathbf{x}\|_{1_{max}} \\
\leq& y(k)
\end{aligned}$$

if $k > t$, and

$$\hat{y}^t(k) = y(k)$$

when $k \leq t$ by definition of $t$-step approximation. □

Hence, our approximation gives a lower bound on $y$ given the last $t$ inputs. This differs from the approximation given in Weyerman and Warnick (2007) as that approximation gave an upper bound on $y$.

Let $P^*$ be the solution to the problem in Eq. 11 and $\hat{P}^{t*}$ the solution to the approximated problem, Eq. 34. We will now construct a bound for the difference between the true cost of using the solution $P^*$, $y(P^*)$, and the true cost of using the solution to the approximated problem, $y(\hat{P}^{t*})$.

**Lemma 9** *Suppose we have the set $\mathscr{A} \subset \mathscr{M}^n$ with a corresponding set of jobs $\{1, 2, \ldots, m\}$ and a quota, $\mathbf{q} \in \mathbb{N}^m$. Then we have the following bound on the error*

$$y(\hat{P}^{t*}) - y(P^*) \leq y(\hat{P}^{t*}) - \hat{y}^t(\hat{P}^{t*}), \tag{37}$$

*where $y(.)$ is the true cost of using the solution, and $\hat{y}^t(.)$ is the approximate cost of using the solution.*

*Proof* Since $\hat{P}^{t*}$ is the optimal solution for the approximated problem,

$$\hat{y}^t(\hat{P}^{t*}) \leq \hat{y}^t(P^*).$$

Similarly, since $P^*$ is the optimal solution for the original problem,

$$y(P^*) \leq y(\hat{P}^{t*}).$$

Because our approximation is a lower bound on $y$,

$$\hat{y}^t(P^*) \leq y(P^*).$$

Combining these inequalities, we get:

$$\hat{y}^t(\hat{P}^{t*}) \leq \hat{y}^t(P^*) \leq y(P^*) \leq y(\hat{P}^{t*}).$$

This ensures that the left hand side of Eq. 37 is non-negative and leads easily to

$$y(\hat{P}^{t*}) - y(P^*) \leq y(\hat{P}^{t*}) - \hat{y}^t(\hat{P}^{t*}).$$

□

This means that when we have a solution, we know the optimal solution is in between the cost as calculated using the approximation and the actual cost of the best sequence of the approximated problem.

We will define the maximum error of the $t$-step approximation much like as in Weyerman and Warnick (2007). Given a set $\mathscr{A}$ and a sequence $P(k-t, k)$, we say that

$$\gamma^t(P(k-t,k)) = \max_{\mathbf{x}} \left\{ \frac{\|A(P(k-t,k)) \otimes \mathbf{x}\|_{1_{max}}}{\|A(P(k-t,k-1)) \otimes \mathbf{x}\|_{1_{max}}}, \frac{\|A(P(k-t,k)) \otimes \mathbf{x}(\ell)\|_{1_{max}}}{\|A(P(k-t,k-1)) \otimes \mathbf{x}(\ell)\|_{1_{max}}} \right\}.$$

From this we define

$$\Gamma^t = \max_P \gamma^t(P).$$

Note that $\Gamma^t$ is another combinatorial optimization problem. This problem is at most as complex as the $t$-step approximation problem. So we assume that if there is enough computational power to solve the $t$-step approximation, then it is possible to calculate $\Gamma^t$.

**Lemma 10** *Given a system as in* Eq. 10 *and a sequence* $P(0,k)$*, for* $k > t$*,* $y(k) \oslash \hat{y}^t(k) \leq \Gamma^t$*. If* $k \leq t$*, then* $y(k) \oslash \hat{y}^t(k) = e$*.*

*Proof* Let a system as in Eq. 10 and a sequence $P(0,k)$ be given. Consider the $t$-step approximation for $k > t$. Then

$$\begin{aligned} y(k) \oslash \hat{y}^t(k) =& \frac{\frac{\|A(P(k-t,k)) \otimes \mathbf{x}(k-t)\|_{1_{max}}}{\|A(P(k-t,k-1)) \otimes \mathbf{x}(k-t)\|_{1_{max}}}}{\frac{\|A(P(k-t,k)) \otimes \mathbf{x}(\ell)\|_{1_{max}}}{\|A(P(k-t,k-1)) \otimes \mathbf{x}(\ell)\|_{1_{max}}}} \\ \leq& \gamma^t(P(k-t,k)) \\ \leq& \Gamma^t. \end{aligned}$$

Now consider the $t$-step approximation for $k \leq t$. Then by Proposition 3

$$y(k) = \hat{y}^t(k)$$

for all $k$. □

These Lemmata lead to the main result of the paper. We will now show that we can calculate a bound on the error of the approximated best solution when compared to the true best solution and that this bound improves as $t$ increases until it reaches 0 at $t = Q$.

**Theorem 6** *If* $\hat{P}^{t*}$ *is the optimal solution to problem* (34)*, then the bound on the error*

$$y(\hat{P}^{t*}) \oslash \hat{y}^t(\hat{P}^{t*}) \leq \bigotimes_{i=t+1}^{Q} \Gamma^t$$

*where*

$$\Gamma^{t+1} \leq \Gamma^t.$$

*Furthermore,*

$$y(\hat{P}^*_Q) \oslash \hat{y}_Q(\hat{P}^*_Q) = 0.$$

*Proof* Suppose $\hat{P}^{t*}$ is the optimal solution to problem (34). Then by Lemma 10, $y(k) \oslash \hat{y}(k) \leq \Gamma^t$ for $k > t$, so

$$\begin{aligned} y(\hat{P}^{t*}) \oslash \hat{y}(\hat{P}^{t*}) =& \bigotimes_{k=0}^{Q} (y(k) \oslash \hat{y}(k)) \\ \leq& \bigotimes_{k=t+1}^{Q} \Gamma^t. \end{aligned}$$

Let $t$ be given. We will let $\tilde{P}_{0,t+1}$ be such that $\gamma^{t+1}(\tilde{P}_{0,t+1}) = \Gamma^{t+1}$. It was shown in Weyerman and Warnick (2007) that for any sequence, $P$, $\gamma^{t+1}(P_{0,t+1}) \leq \gamma^t(P_{0,t})$, so

$$\begin{aligned} \gamma^{p+1}(\tilde{P}_{0,t+1}) \leq& \gamma_t(\tilde{P}_{0,t}) \\ \leq& \Gamma^t. \end{aligned}$$

The final statement of the theorem follows since for $k \leq t$, the approximation is exact. So if $t = Q$, then the approximation is exact up to $k = Q$. □

## 7 Conclusion

We have presented a model approximation method for batch flowshop scheduling to minimize makespan. This problem is shown to be $\mathcal{NP}$-complete, which motivates the need for an approximation of the system. We have shown that this system can be modeled as a discrete event system in the max-plus algebra. Furthermore, these systems represent a subset of a class of stable systems in the max-plus algebra. Due to this stability, we present a method of model approximation that reduces the horizon of the optimization problem. This model approximation method, which gives a lower bound to the actual output of the system, scales to the amount of computational power available and offers a bound on the error between the obtained solution and the optimal solution. This bound improves as the model approximation is refined and the bound eventually reaches zero.

**Acknowledgments** This work has been funded by generous contributions from the Department of Homeland Security (DHS) Science and Technology Directorate, Homeland Security Advanced Research Projects Agency (HSARPA), Cyber Security Division (DHS S&T/HSARPA/CSD), LRBAA 12-07 via contract number HSHQDC-13-C-B0052, as well as from contributions from AFRL FA8750-09-2-0219 and ATK Thiokol. All material in this paper represents the position and work of the authors and not necessarily that of the sponsors.

The authors would like to thank Professor Tyler Jarvis of the Mathematics Department at Brigham Young University for useful discussions during the early phase of this research. We also gratefully acknowledge the considerable work and suggestions of the anonymous reviewers which, in our opinion, contributed significantly to the readability and clarity of this work.

## Appendix: Proof of Theorem 1

**Lemma 11** *Given $B \in \mathbb{R}_{\max}^{n\times n}$ that satisfies* Eq. 7 *and $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$, the matrix resulting from finitely many left multiplications of $P_i(k)$ and $R_i(k)$ for any $i$ with $B$ satisfies* Eq. 7.

*Proof* Let $B \in \mathbb{R}_{\max}^{n\times n}$ that satisfies Eq. 7, $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$, and $i \leq n$ be given. After a left multiply of $P_i(k)$, Eq. 7 is trivially satisfied. Suppose that $i < n$. Consider the product $A = R_i(k) \otimes B$. Because the only difference between $A$ and $B$ is in rows $i$ and $i+1$, we want to show that $a_{ij} \geq a_{i,j+1}$ and $a_{i+1,j} \geq a_{i+1,j+1}$ for all $j < n$. We will consider two cases.

Suppose $b_{ij} \geq b_{i+1,j}$. Then $a_{ij} = b_{ij} = a_{i+1,j}$. Because $B$ satisfies (7) it follows that $b_{ij} \geq b_{i,j+1}$ and $b_{ij} \geq b_{i+1,j} \geq b_{i+1,j+1}$. So $a_{ij} \geq a_{i,j+1}$ and $a_{i+1,j} \geq a_{i+1,j+1}$.

Suppose $b_{i+1,j} \geq b_{ij}$. We achieve the same result as the previous case in the same manner.

Thus, because $B$ satisfies Eq. 7 after a left multiply of an arbitrary $P_i$ or $R_i$, after a finite number of left multiplies, the resulting matrix will satisfy Eq. 7. □

**Lemma 12** *Given $B \in \mathbb{R}_{\max}^{n\times n}$ that satisfies* Eq. 8 *and $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$, the matrix resulting from finitely many left multiplications of $P_i(k)$ and $R_i(k)$ for any $i$ with $B$ satisfies* Eq. 8.

*Proof* Let $B \in \mathbb{R}_{\max}^{n\times n}$ that satisfies Eq. 8, $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$, and $i \leq n$ be given. Left multiplying $B$ by $P_i$ adds the same number to every element of row $i$. As this does not change $\xi_{ij}$ for any

$j$, the resulting matrix still satisfies Eq. 8. Now suppose that we left multiply $B$ by $R_i$ for some $i < n$. The resulting matrix, $A$, has rows $i$ and $i+1$ equal. We now have

$$\begin{aligned}\xi_{ij}(A) &= a_{ij} - a_{i,j+1}\\ &= \max(b_{ij}, b_{i+1,j}) - \max(b_{i,j+1}, b_{i+1,j+1})\\ &= a_{i+1,j} - a_{i+1,j+1}\\ &= \xi_{i+1,j}(A).\end{aligned}$$

We need now show that $\xi_{i-1,j}(A) \geq \xi_{ij}(A)$ and that $\xi_{i+1,j}(A) \geq \xi_{i+2,j}(A)$. We will handle each of four cases separately.

– Suppose $b_{ij} \geq b_{i+1,j}$ and $b_{i,j+1} \geq b_{i+1,j+1}$. Then

$$a_{ij} - a_{i,j+1} = b_{ij} - b_{i,j+1}$$

and

$$\begin{aligned}a_{i+1,j} - a_{i+1,j+1} &= b_{ij} - b_{i,j+1}\\ &\geq b_{i+1,j} - b_{i+1,j+1}.\end{aligned}$$

– Suppose $b_{ij} \geq b_{i+1,j}$ and $b_{i,j+1} \leq b_{i+1,j+1}$. Then

$$\begin{aligned}a_{ij} - a_{i,j+1} &= b_{ij} - b_{i+1,j+1}\\ &\leq b_{ij} - b_{i,j+1}\end{aligned}$$

and

$$\begin{aligned}a_{i+1,j} - a_{i+1,j+1} &= b_{ij} - b_{i+1,j+1}\\ &\geq b_{i+1,j} - b_{i+1,j+1}.\end{aligned}$$

– Suppose $b_{ij} \leq b_{i+1,j}$ and $b_{i,j+1} \geq b_{i+1,j+1}$. But then

$$b_{ij} - b_{i,j+1} \leq b_{i+1,j} - b_{i+1,j+1}$$

which violates the assumption on $B$, so this case is not possible.

– Finally, suppose $b_{ij} \leq b_{i+1,j}$ and $b_{i,j+1} \leq b_{i+1,j+1}$. Then

$$\begin{aligned}a_{ij} - a_{i,j+1} &= b_{i+1,j} - b_{i+1,j+1}\\ &\leq b_{i,j} - b_{i+1,j}\end{aligned}$$

and

$$a_{i+1,j} - a_{i+1,j+1} = b_{i+1,j} - b_{i+1,j+1}.$$

We have shown that in any case, $\xi_{ij}(A) \leq \xi_{ij}(B)$ and $\xi_{i+1,j}(A) \geq \xi_{i+1,j}(B)$. For any $l \neq i$ and $l \neq i+1$, $\xi_{lj}(A) = \xi_{lj}(B)$ for all $j$. Thus, $\xi_{i-1,j}(A) \geq \xi_{ij}(A) = \xi_{i+1,j}(A) \geq \xi_{i+2,j}(A)$, so Eq. 8 is satisfied after a left multiply of $R_i$.

Because we have shown that the resulting matrix after a left multiply of arbitrary $P_i$ or arbitrary $R_i$ satisfies Eq. 8, it follows that Eq. 8 is satisfied after a finite number of left multiplies. □

We are now ready to prove the main theorem.

*Proof of Theorem 1* Let $(\mathbf{c}, \boldsymbol{\tau})(\alpha)$ be given. The algorithm for constructing $A(\alpha)$ begins with $I_{max}$. The block of $I_{max}$ consisting of $i_{11}$ is trivially in $\mathscr{M}^n$. Thus we know by the Lemmata 11 and 12 that this block satisfies Eq. 7 and Eq. 8 throughout the algorithm. We

will show by induction that by the end of the algorithm the entire matrix satisfies both of these. Suppose that at the $p^{th}$ stage of the algorithm, the block from $b_{11}$ to $b_{kk}$ of the current matrix $B$ satisfies Eq. 7 and Eq. 8 and the remainder of the matrix consists of $\epsilon$ off the diagonal and $e$ on the diagonal. Consider now $C = R_k \otimes B$, we can suppose without loss of generality that this is the next operation the algorithm performs. Row $k+1$ of $C$ is equal to row $k$ of $C$, with the only difference between row $k$ of $C$ and row $k$ of $B$ being that $c_{k,k+1} = e$ and $b_{k,k+1} = \epsilon$. Therefore $\xi_{ik}(C) = \infty$ for $i < k$ and $\xi_{kk}(C) < \infty$, so the block from $c_{11}$ to $c_{k+1,k+1}$ satisfies (8). Also, because we must multiply by $P_k$ before $R_k$, $c_{kk} > c_{k,k+1} = 0$, so the block from $c_{11}$ to $c_{k+1,k+1}$ satisfies Eq. 7. The remaining rows and columns of $C$ consist of $\epsilon$ off the diagonal and $e$ on the diagonal. Therefore, by induction, $A$ must satisfy Eq. 7 and Eq. 8.

It remains to be shown that $A$ satisfies Eq. 6 and Eq. 9. By the algorithm, we can write,

$$A = R_{n-1} \otimes B_{n-1} \otimes R_{n-2} \otimes B_{n-2} \otimes \ldots \otimes R_1 \otimes P_1 \otimes I_{max}$$

where $B_{n-j}$ is some intermediate matrix sum. This sequence of $R_i$'s shows that the 0 entries on the diagonal or $I_{max}$ will be cascaded down through the columns of the matrix, it will also ensure that the entries just above the diagonal are greater than $-\infty$, and we see that $A$ satisfies Eq. 9.

For Eq. 6, consider the last multiplication of $P_i$ for some $i$. This is necessarily followed by a multiplication of $R_i$, at this point the resulting matrix, $C$, has $c_{ij} = c_{i+1,j}$. This matrix is necessarily multiplied by $P_{i+1}$ before another $R_i$ or $P_i$; the resulting matrix, which we will call $D$, has $d_{ij} \leq d_{i+1,j}$, at this point we know $a_{ij} = d_{ij}$ and $a_{i+1,j} \geq d_{i+1,j}$, so $A$ must satisfy Eq. 6. □

## References


Ahmadi JH, Ahmadi RH, Dasu S, Tang CS (1992) Batching and scheduling jobs on batch and discrete processors. Oper Res 39(4):750–763

Beck C, Doyle J, Glover K (1996) Model reduction of multidimensional and uncertain systems. IEEE Trans Autom Control 41(10):1466–1477

Baccelli F, Cohen G, Olsder GJ, Quadrat JP (1992) Synchronization and linearity. Wiley

Bemporad A (2003) Multiparametric nonlinear integer programming and explicit quantized optimal control. In: Proceedings 42nd IEEE conf decis control. pp 3167–3172

Blömer F, Günther H (1998) Scheduling of a mutli-product batch process in the chemical industry. Comput Ind 36:245–259

Boccadoro M, Valigi P (2003) A modelling approach for the dynamic scheduling problem of manufacturing systems with non negligible setup times and finite buffers. In: Proceedings 42nd IEEE conf decis control

Bouquard JL, Lenté C, Billaut JC (2006) Application of an optimization problem in max-plus algebra to scheduling problems. Discret Appl Math 154(15):2064–2079

Bouquard JL, Lenté C, Billaut JC (2006) Two-machine flow shop scheduling problems with minimal and maximal delays. 4OR: A Q J Oper Res 4:15–28

Cheng T, Ng C, Yuan J, Liu Z (2004) Single machine parallel batch scheduling subject to precedence constraints. Nav Res Logist 51:949–958

Cohen G, Dubois D, Quadrat JP, Viot M (1985) A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. IEEE Trans Autom Control AC-30(3):210–220

Corona D, Giua A, Seatzu C (2005) Quantized optimal control of discrete-time systems. In: 10th IEEE conference of ETFA

Deng X, Feng H, Li G, Lui G (2002) A PTAS for minimizing total completion time of bounded batch scheduling. Int J Found Comput Sci 13(6):817–827

Dobson G, Nambimadom RS (2001) The batch loading and scheduling problem. Oper Res 49(1):52–65
Dullerud G, Paganini F (2000) A course in robust control theory: a convex approach. Appl Math. Springer
Dupont L, Dhaenens-Flipo C (2002) Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. Comput Oper Res 29(7):807–819
French S (1982) Sequencing and scheduling: an introduction to the mathematics of the job-shop. Math Appl. Ellis Horwood Limited
Gaubert S, Mairesse J (1999) Modeling and analysis of timed petri nets using heaps of pieces. IEEE Trans Autom Control 44(4):683–697
Glass C, Potts C, Strusevich V (2001) Scheduling batches with sequential job processing for two-machine flow and open shops. INFORMS J Comput 13(2):120–137. Spring
Gupta JND (1986) Flowshop schedules with sequence dependent setup times. J Oper Res Soc Jpn 29(3):206–219
Hall N, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44(3):510–525
Heidergott B, Olsder GJ, van der Woude J (2006) Max plus at work: modeling and analysis of synchronized systems: a course on max-plus algebra and its applications princeton series in applied mathematics. Princeton University Press
Hochbaum D, Landy D (1997) Scheduling semiconductor burn-in operations to minimize total flowtime. Oper Res 45(6):874–885
Jordan C, Drexl A (1998) Discrete lotsizing and scheduling by batch sequencing. Manag Sci 44(5):698–713
Kondili E, Pantelides CC, Sargent RWH (1993) A general algorithm for short-term scheduling of batch operations - I. MILP formulation. Comput Chem Eng 17(2):211–227
Lin B, Cheng T (2001) Batch scheduling in the no-wait two-machine flowshop to minimize the makespan. Comput Oper Res 28:613–624
López-Mellado E, Villanueva-Paredes N, Almeyda-Canepa H (2005) Modelling of batch production systems using Petri nets with dynamic tokens. Comput Oper Res 67(6):541–558
Parker RG (1995) Deterministic scheduling theory. Chapman and Hall
Pinedo ML (2005) Planning and scheduling in manufacturing and services. Springer series in operations research. Springer Science+Business Media Inc.
Reddi S, Ramamoorthy C (1972) On the flow-shop sequencing problem with no wait in process. Oper Res Q 23(3):323–331
Rich SH, Prokopakis GJ (1986) Scheduling and sequencing of batch operations in a multipurpose plant. Ind Eng Chem Process Des Dev 25:979–988
Riera D, Narciso M, Benqlilou C (2005) A petri nets-based scheduling methodology for multipurpose batch plants. Simul 81:613–623
Roe B, Papageorgiou L, Shah N (2005) A hybrid MILP/CLP algorithm for multipurpose batch process scheduling. Comput Chem Eng 29:1277–1291
Sand G, Engell S (2004) Modeling and solving real-time scheduling problems by stochastic integer programming. Comput Chem Eng 28:1087–1103
Sanmartí E, Puigjaner L, Holczinger T, Friedler F (2002) Combinatorial framework for effective scheduling of multipurpose batch plants. AIChE J 48(11):2557–2570
Savkin A (2003) Optimal distributed real-time scheduling of flexible manufacturing networks modeled as hybrid dynamical systems. In: Proceedings IEEE conf decis control
Su R, Woeginger G (2011) String execution time for finite languages: max is easy, min is hard. Autom 47(10):2326–2329
Van den Boom TJJ, De Schutter B (2006) Modelling and control of discrete event systems using switching max-plus-linear systems. Control Eng Pract 14(10):1199–1211
van Eekelen J, Lefeber E, Rooda J (2006) Feedback control of 2-product server with setups and bounded buffers. In: Proceedings IEEE American control conference
van Eekelen J, Lefeber E, Roset B, Rooda J (2005) Control of manufacturing systems using state feedback and linear programming. In: Proceedings of IEEW conference on decision and control
Vanderbei RJ (2001) Linear programming: foundations and extensions, 2nd edn. Kluwer Academic Publishers
Weyerman W, Warnick S (2007) Monotonically improving error bounds for a sequence of approximations for makespan minimization of batch manufacturing systems. In: Proceedings of IEEE conference on decisions and control
Yajima T, Ito T, Hashizume S, Kurimoto H, Onogi K (2004) Control of batch processes based on hierarchical petri nets. IEICE Trans Fundam E87-A(11):2895–2904
Yurdakul M, Odrey NG (2004) Development of a new dioid algebraic model for manufacturing with the scheduling decision making capability. Robot Auton Syst:207–218

**Sam Weyerman** received the BS and MS in computer science from Brigham Young University in 2005 and 2008, respectively. He currently works for Google, where his interests span data analysis and large scale systems.

**Anurag Rai** received his BS and MS degrees in Computer Science from Brigham Young University in 2010 and 2012, respectively. Currently he is pursuing a PhD in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include optimization, stochastic control, and control theory.

**Sean Warnick** received the B.S.E. degree from Arizona State University in 1993, and the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology in 1995 and 2003, respectively. He is currently an Associate Professor of Computer Science at Brigham Young University, where he directs the Information and Decision Algorithms Laboratories. His research is in the representation, identification, and control of distributed and networked systems.